ニューラルネット

本稿は生物神経を分析し、どのように生物神経と同じように活動する人工ニューロンを作成するかを仮説する。また、誤差逆伝播法というニューラルネットの学習法を詳細に分析し、各学習法の価値や工学的応用などを解説する。

関西外国語大学

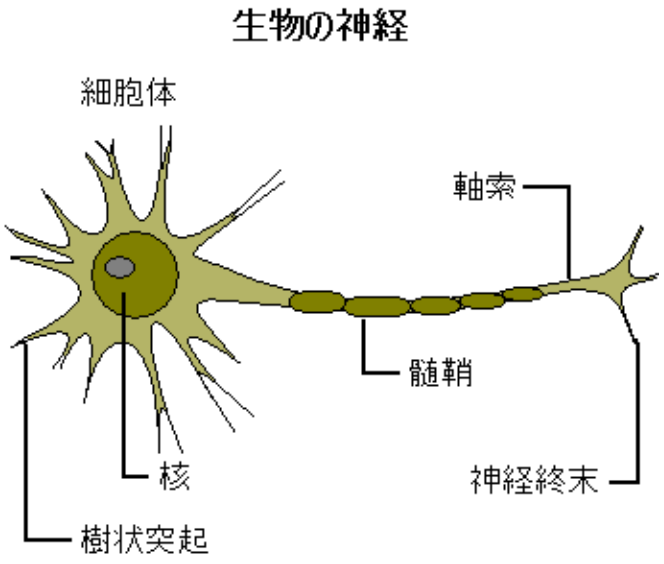
Kenneth（Kenny）Cason

平成２００８年３月１日

１．神経と神経回路網とは

各生物には神経という細胞があり、神経は情報伝達の役割を担う。神経もニューロンと呼ばれる。神経は生物にとって大事なもので、様々な種類の神経細胞がある。中枢と末梢の神経に大別され、また構造的・機能的な観点により分類される。例えば、機能的に運動神経と知覚神経は異なる。運動神経は体および内臓を運動させるが、知覚神経は環境からデータを知覚し、他の関連している神経に信号を伝達する。更に内臓の運動・知覚に関係するものは自律神経としてまとめられる。自律神経は交感神経と副交感神経に分けられる。神経は構造的に異なるが、全神経は機能的に類似している部分がある。まず、神経細胞の中心は普通の細胞と同じように、神経核、小胞体またゴルジ体などがある。神経細胞は多数の樹状突起を持ち、回りの関連している神経細胞から運ばれる情報が電位として突起に伝達される。一つの神経細胞内の膜電位が十分な程度（閾値）に達すると、電気的変化が細胞膜の微細構造的変化を起こし、活動電位が発生する。図１は一般の生物の神経を表している（Wikipedia「神経」）。

図１



一つの神経だけでは生物が思想や脳のような機能などする事が出来ない。更に、図２によると、世界中の一番単純な生物を分析しても、虫螻の神経数は約３０２で、人間は約１

０００億である。また一つの神経と他の神経と関連している部分はシナプスという。生物の知識はシナプスの結合のパターンに保存されているので、高次思想にとって神経の数より、シナプスの数の方が大切である。一つの神経が回りの神経と関連するものは一般的に神経回路網と呼ばれ、生物の神経回路網を模倣するモデルはよくニューラルネットと呼ばれる。図２を分析することで、神経とシナプスの数の割合が分かる。シナプスの数は神経の数より大分多く、人間のデータを見れば、子供の神経数の方が大人の神経数より多いことが分かる。生物は生まれる時、シナプスの数が一番多く、年を取れば取る程学習する事で、使われないシナプスや大切ではないシナプスなどが消え、大切なシナプスが強調される。従って、生物と同じように、ニューラルネットが作られる時から、学習をする必要があり、最適な学習法も選択した方が良い。しかし、現在の技術にはニューラルネットは様々な応用があり、応用によって、各学習法は一長一短である。それゆえ、次に本論は生物の神経回路を模倣するモデルの作成法を分析し、よく使われる学習法も分析し、応用による最適なニューラルネットと学習理論を考察する。例えば、人工知能とパターン認識ソフトの最適なニューラルネットの構成と学習法は異なる。しかし、現実的に人間のような進化された生物の脳は複雑で、より単純な生物の神経数と比較すると、神経数とシナプス数の差が非常に多い。そのため、神経数が低い生物の脳の方が模倣し易い。

図２

動物名	神経数	シナプス数
虫螻	302	5600
蟻	10,000	–
ゴキブリ	1,000,00	–
鼠	10^7 1000 万	$8x10^{10}$ 8000 万
鮹	3000 万~1 億	–
人間 （大人）	10^{12} 1000 億	10^{14} – $5x10^{14}$ 100 兆~500 兆
人間 （子供）	10^{12} 1000 億	10^{16} 1 京

象	>2000 億	-
白長須鯨	>2000 億	-

（Wikipedia「List of Animals by number of Neurons」）

２．ニューラルネットのデザイン

まず、生物の神経回路の基本的な単位（生物神経）を分析し、人工ニューロンという生物神経を模倣するモデルを作成する。前の表２を見れば、生物神経はシナプスが非常に多く、計算的にコンピュータはエネルギーおよび時間を用い過ぎる。ゆえに、シナプスの数で二つの神経の関連の強さを表すより、二つのニューロンの関連の強さは結合荷重（W）で表す。この方が計算的に単純で、結果も大抵同じである。生物神経のように情報を伝達する為、ニューロンは入力信号を全部足して、閾値関数に入れる。例えば、図３の閾値関数は各入力信号とその対応するシナプスの荷重を乗法し、最後に全ての値を加法し、その合計（*a*)は特有な限度を超えると、ニューロンが他の関連しているニューロンに出力信号を運ぶ。

図３

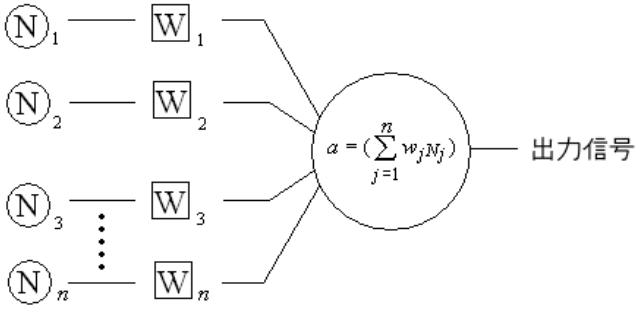


図４

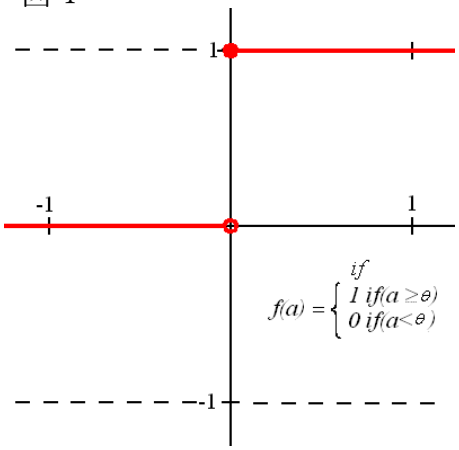
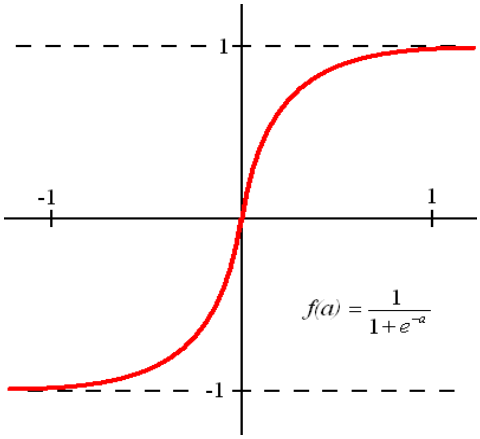


図５



次に、閾値関数を詳しく分析してみよう。二つの種類のよく使われる関数があり、図４および図５で表れている。図４のような閾値関数で（*a*）が０地点に到達すると、１になり、ニューロンが出力信号を運び、また０に戻る。しかし、生物神経はこのような関数がなく、図５のような方式で使われている。図５の関数でニューロンは緩やかなカーブで徐々に上がっていき、また特有な限度を超えると、ニューロンが表４と同じ活動をする。

生物の神経回路の結合には方向性と強度がある。生物の神経回路は特に複雑で、計算するのは非常に困難である（図７）。計算し易くなるようにニューラルネットの人工ニューロンは層に分けられる。一般のニューラルネットは三つの層に分けられ、入力層、中心層、出力層である（図６）。中心層は外から直接に観察出来ないので、隠れ層とよく呼ばれる。入力層および出力層はそれぞれ一つの層だけであるが、ニューラルネットによると、中心は多数の層がある可能性である。ニューラルネットの中心層の数が増加すればする程、学習する時間も急速に増加するが、そのニューラルネットの学習能力も増加する。次に、どのような方法でニューラルネットを学習するのかを分析してみよう。

図６

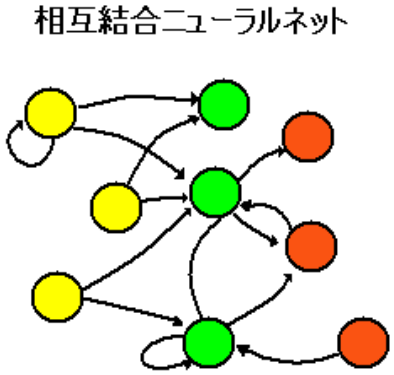
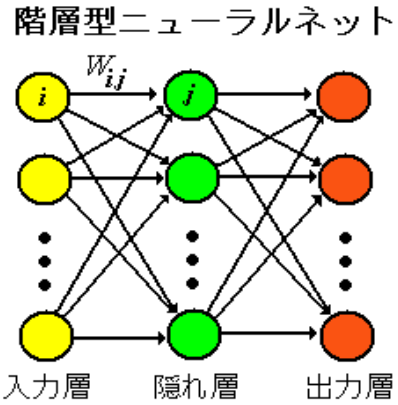


図７



３．ニューラルネットの学習法

ニューラルネットのデザインを決定した後、どの学習法を使用するかを選択する。ニューラルネットの学習法は「教師あり学習（図８）」と「教師なし学習（図９）」に大別される。教師あり学習法ではニューラルネット（NN）の実際の出力と教師信号を比較し、何か

のアルゴリズム（算法）で差を小さくする。差が十分に小さくなるまで繰り返す。一方で、教師なし学習法では内部的なメカニズムで学習する。そして、この教師なし学習を用いているロボットは完全に主動的に動く。

図８

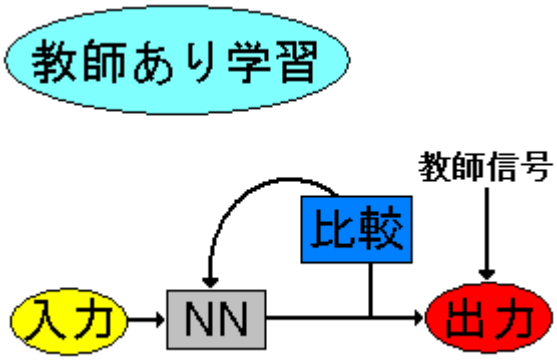
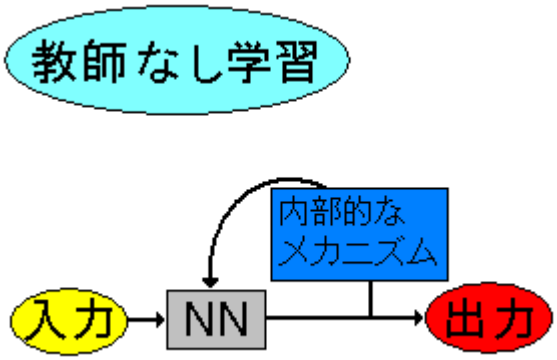


図９



人間の脳は教師あり学習で学習し、最終的な目標は人工知能であるので、本稿は教師なし学習より、教師あり学習に集中する。様々な教師ある学習法があるので、よく使われる学習法を記述したいと思う（**浅井潔**）。

４．パーセプトロンの学習アルゴリズム

パーセプトロンの学習とは、ニューラルネットの教師あり学習法の一つで、ニューラルネットの出力値と教師信号の誤差を最小化する学習法である。まず、値を下に定義する。

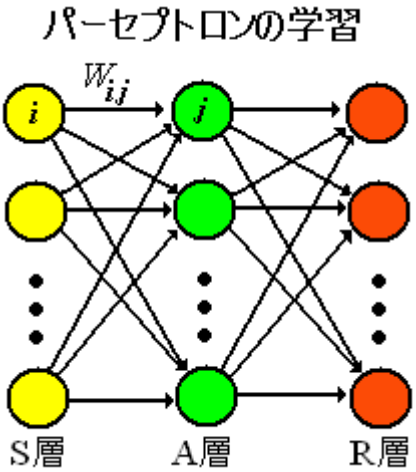
入力　x $x=(x_1, x_2, \ldots, x_m, -1)$　　結合荷重　$w_j=(w_{1j}, w_{2j}, \ldots, w_{mj}, \theta_j)$

出力　$y_j=H[w_j \cdot x]$　　教師信号　d_j

そして、ニューラルネットが学習するまで、下の四つのステップを繰り返す（**浅井**）。

1. w_jの初期値（$\neq 0$）をランダムな値に決める。
2. 学習のデータセットの１データ $x^s=(x_1^s, x_2^s, \ldots, x_m^s, -1)$ を入力として出力 $y_j^s=H[w_j \cdot x^s]$ を計算する。
3. 対応する教師信号 d_j^s を用いて w_j を更新する。 $w_j \leftarrow w_j+(d_j^s-y_j^s)x^s$
4. ステップ２.に戻って出力と教師信号の差が最小化するまで繰り返す。

表１０



５．誤差逆伝播法（バックプロパゲーション）

誤差逆伝播法は、ニューラルネットを訓練する為に使われる教師あり学習である。生物の脳が働く時、磁気共鳴画像では誤差逆伝播法が行う事が観察出来たので、誤差逆伝播法と他の変化形はよく学習法として使われる。

誤差逆伝播法のアルゴリズム（Wikipedia「バックプロパゲーション」）：

1. ニューラルネットに学習の為のデータを入力する。
2. ニューラルネットの出力とそのデータの最適解を比較し、各出力ニューロンについて誤差を計算する。
3. 各ニューロンの期待される出力値と倍率、また要求された出力と実際の出力の差を計算する。この差は局所誤差と言う。
4. 各ニューロンの結合荷重を局所誤差が小さくなるように調整する。
5. より大きな結合荷重で接続された前段のニューロンに対し、局所誤差の責任があると判定する。
6. そのように判定された前段のニューロンの更に前段のニューロン群について同様の処理を行う。

次に、どのようにニューロンの結合荷重を更新するかをもっと詳しく説明する。まず、誤差を計算する式の値を定義する。入力層ユニット i →中間層ユニット j の結合荷重を

$w^h_{j,i}$、中間層ユニット j →出力層ユニットの出力値を y、教師信号を t とする。次に、出力の誤差 E を二乗誤差で下の式で表れている。

$$\delta = \frac{1}{2}(t-y)^2$$

＊この誤差 δ が０になれば、実際の出力は期待された出力と同じという事になり、誤差逆伝播法のアルゴリズムが終了である。

そして、計算した誤差 E を使用して、各ニューロンの結合荷重を更新する。 ε は更新の割合を表す学習係数である。更新式は以下の微分計算が定義される。

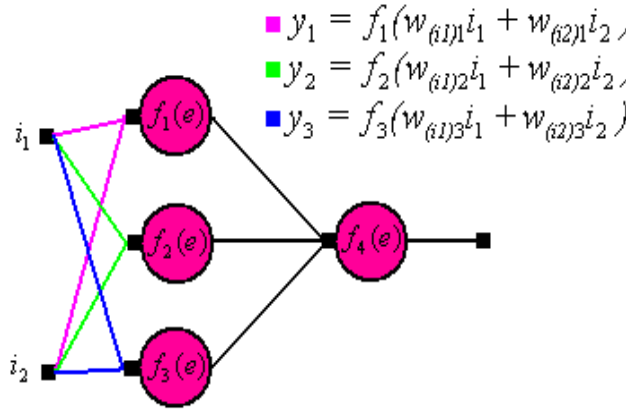
$$\Delta w^o_{k,j} = -\varepsilon \frac{\partial \delta}{\partial w^o_{k,j}}$$
$$w^o_{k,j} \leftarrow w^o_{k,j} + \Delta w_{k,j}$$
$$\Delta w^h_{j,i} = -\varepsilon \frac{\partial \delta}{\partial w^h_{j,i}}$$
$$w^h_{j,i} \leftarrow w^h_{j,i} + \Delta w^h_{j,i}$$

以上の定義された式で、中間層→出力層の結合荷重の更新の情報を入力層→中間層の結合荷重の更新に利用することになり、誤差が逆方向（出力層から）に伝播しながら、結合荷重を調整する（小笠原）。

以下の図で誤差逆伝播法を段階的に説明する。

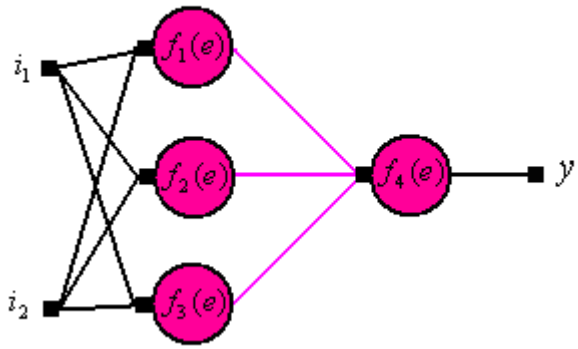
η ＝学習係数、 δ ＝誤差、 y ＝出力、 $f_I(e)$ ＝閾値関数、w＝結合荷重、 i ＝入力データ

まず、データを入力し、中間層のニューロンの値を計算する。

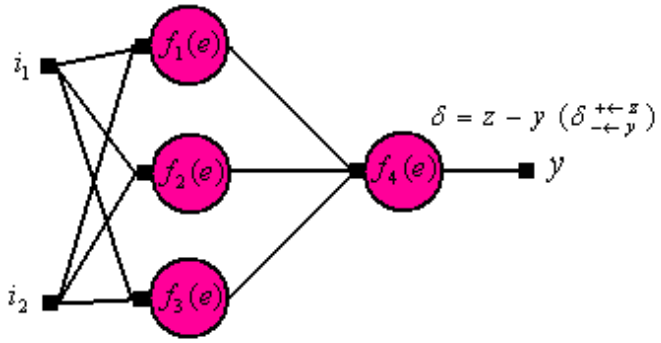


出力層のニューロンの値を計算する。

■ $y = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$

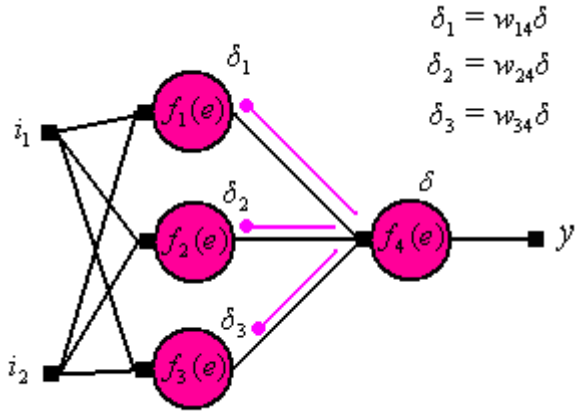


実際の入力と期待されている値を比較し、誤差を計算する。



逆に伝播し、中間層の各結合荷重の誤差を計算する。

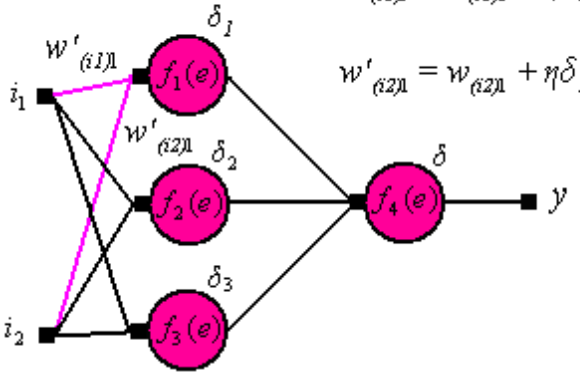
$\delta_1 = w_{14}\delta$
$\delta_2 = w_{24}\delta$
$\delta_3 = w_{34}\delta$



入力層の結合荷重の誤差を計算する。

$$w'_{(11)1} = w_{(11)1} + \eta\delta_1 \frac{df_1(e)}{de} i_1$$

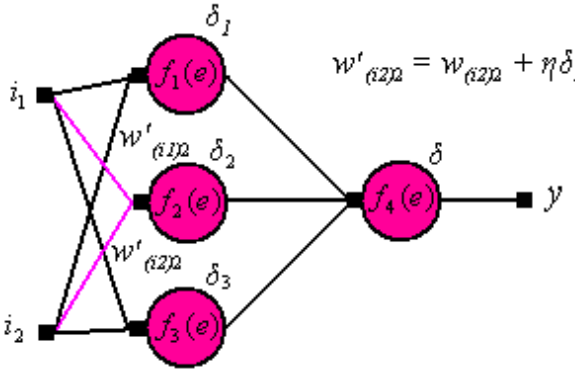
$$w'_{(12)1} = w_{(12)1} + \eta\delta_1 \frac{df_1(e)}{de} i_2$$



次は、各結合荷重を調整する。

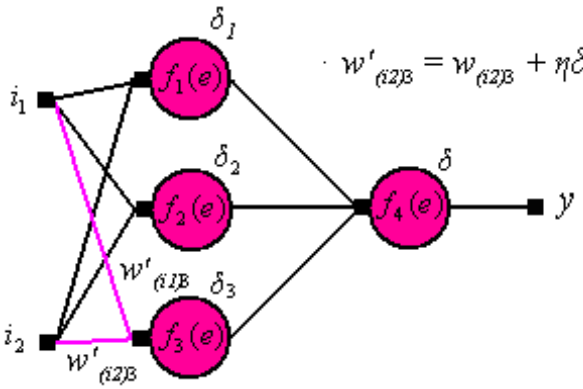
$$w'_{(11)2} = w_{(11)2} + \eta\delta_2 \frac{df_2(e)}{de} i_1$$

$$w'_{(12)2} = w_{(12)2} + \eta\delta_2 \frac{df_2(e)}{de} i_2$$



$$w'_{(11)3} = w_{(11)3} + \eta\delta_3 \frac{df_3(e)}{de} i_1$$

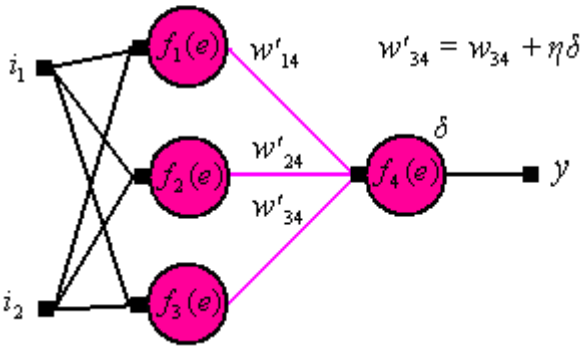
$$w'_{(12)3} = w_{(12)3} + \eta\delta_3 \frac{df_3(e)}{de} i_2$$



$$w'_{14} = w_{14} + \eta\delta \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta\delta \frac{df_4(e)}{de} y_3$$



６．誤差逆伝播法でニューラルネットを学習する結果

次、本稿は誤差逆伝播法でニューラルネットを学習し、学習率や中間層の神経数によって、どのように最適にニューラルネットを学習するかを仮説する。まず、ニューラルネットの学習の結果をもっと簡単に分析する為、単純なデータを学習した方が良い。それゆえ、ニューラルネットに論理積を学習する。図 6.1 は論理積の真理値表である。

図6.1

論理積(AND)		
命題p	命題q	p∧q
偽	偽	偽
偽	真	偽
真	偽	偽
真	真	真

論理積を学習するニューラルネットを作成する為、入力が二つ(命題pと命題q)で、出力が一つ(p∧q)である。最初の実験として、中間層数を一つ、中間層のニューロン数を二つに設定し、学習率と誤差の値を変更して、ニューラルネットが学習回数を観察する。以下の図はその訓練のデータを表している。

図6.2

誤差逆伝播法でANDを学習する結果				
中間層数	中間層ノード数	学習率(教師信号)	誤差	平均学習回数
1	2	0.1	0.1	844
1	2	0.1	0.01	2688
1	2	0.1	0.001	9685
1	2	0.1	0.0001	38063
1	2	0.2	0.1	407
1	2	0.2	0.01	1250
1	2	0.2	0.001	4543
1	2	0.2	0.0001	29578
1	2	0.3	0.1	289
1	2	0.3	0.01	740
1	2	0.3	0.001	2768
1	2	0.3	0.0001	19915
1	2	0.4	0.1	241
1	2	0.4	0.01	555
1	2	0.4	0.001	9507

1	2	0.4	0.0001	11953
1	2	0.5	0.1	157
1	2	0.5	0.01	425
1	2	0.5	0.001	1595
1	2	0.5	0.0001	10269

図 6.2 から誤差が小さくなればなる程、平均学習回数が指数的に増加し、値が高い学習率の平均学習回数は値が低い学習率のより高いことが分かる。しかし、図 6.3 を見れば、学習率が高過ぎると、学習回数が非常に増加し、学習出来ない時もある。

次、以上のニューラルネットと同じように中間層のニューロン数を三つに設定し、学習してみよう。図 6.3 はこの結果を表している。

図 6.3

誤差逆伝播法で AND を学習する結果				
中間層数	中間層ノード数	学習率 (教師信号)	誤差	平均学習する必要の数
1	3	0.1	0.1	672
1	3	0.1	0.01	2035
1	3	0.1	0.001	7819
1	3	0.1	0.0001	52187
1	3	0.2	0.1	266
1	3	0.2	0.01	997
1	3	0.2	0.001	4174
1	3	0.2	0.0001	23568
1	3	0.3	0.1	191
1	3	0.3	0.01	659
1	3	0.3	0.001	2504
1	3	0.3	0.0001	15789
1	3	0.4	0.1	147
1	3	0.4	0.01	509
1	3	0.4	0.001	1847
1	3	0.4	0.0001	15233

1	3	0.5	0.1	116
1	3	0.5	0.01	394
1	3	0.5	0.001	1601
1	3	0.5	0.0001	9215

このニューラルネットの学習の結果は殆ど同じだか、中間層のニューロンが三つのニューラルネットの場合はより速く学習した。

次、学習率だけを変更する実験をすることで、学習率の限定が分かる。例えば、具体的にあるニューラルネットの最大設定できる学習率が分かる。結果は図6.4で表れている。このニューラルネットの場合で、学習率が３０以上になれば、学習回数が非常に高くなり、学習出来ない時も沢山あった。人間と比較すると、学習率の値は処罰の強さと同じで、人間の処罰が強過ぎれば学ばなく、一方で処罰が弱過ぎれば学ぶのは遅く、学ばない可能性もある。

図6.4

誤差逆伝播法でANDを高い学習率を使って学習する結果				
中間層数	中間層ノード数	学習率(教師信号)	誤差	平均学習回数
1	3	0.1	0.0001	52187
1	3	0.2	0.0001	23568
1	3	0.3	0.0001	157789
1	3	0.4	0.0001	15233
1	3	0.5	0.0001	9215
1	3	0.75	0.0001	6261
1	3	1.0	0.0001	4701
1	3	2.0	0.0001	2183
1	3	4.0	0.0001	1293
1	3	10.0	0.0001	415
1	3	15.0	0.0001	302
1	3	20.0	0.0001	294
1	3	25.0	0.0001	273
1	3	30.0	0.0001	80143
1	3	30.0より以上	0.0001	学習出来ない

ニューラルネットの様子をもっと詳細に理解出来るように、中間層数と中間層ノード数も変更すれば中間層数はどのように学習回数を影響することがわかる。図6.5はこの結果を表している。

図6.5

中間層数と中間層のノード数を変更して、誤差逆伝播法でANDを学習する結果				
中間層数	中間層ノード数	学習率(教師信号)	誤差	平均学習回数
1	2	0.5	0.0001	10518
1	3	0.5	0.0001	10145
1	4	0.5	0.0001	9093
1	5	0.5	0.0001	8349
1	10	0.5	0.0001	7486
2	2	0.5	0.0001	9004
2	3	0.5	0.0001	6714
2	4	0.5	0.0001	6210
2	5	0.5	0.0001	4462
2	10	0.5	0.0001	4209
3	2	0.5	0.0001	9369
3	3	0.5	0.0001	7647
3	4	0.5	0.0001	5820
3	5	0.5	0.0001	4253
3	10	0.5	0.0001	3353
4	2	0.5	0.0001	30887
4	3	0.5	0.0001	17189
4	4	0.5	0.0001	14102
4	5	0.5	0.0001	6571
4	10	0.5	0.0001	3147
10	5	0.5	0.0001	学習出来ない
10	10	0.5	0.0001	学習出来ない
10	15	0.5	0.0001	学習出来ない

10	20	0.5	0.0001	学習出来ない
10	25	0.5	0.0001	学習出来ない
10	50	0.5	0.0001	1835

図6.5によって、中間層数の影響は少し分かり難くいが、中間層数と中間層のノード数が同時に上がるにつれて、平均学習回数が下がることが分かる。更に、中間層数が１０の場合、ニューラルネットは学習出来なくなってしまった。しかし、ノードが高ければ学習出来るようになり、平均学習回数も比較的に低い。

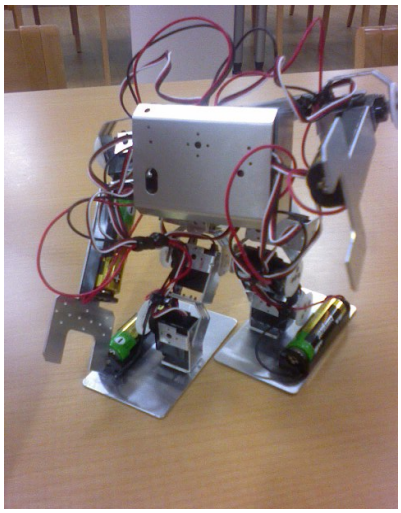
ニューラルネットを学習する時、学習率や中間層のノード数や中間層数などが学習回数に影響があり、学習データや期待されている出力によって、最適な学習法も異なる。後、本稿の全ての実験は誤差逆伝播法を使用したが、応用によって、他の学習法も価値がある。誤差逆伝播法以外の学習法を使用するとニューラルネットの様子についての知識が蓄積する。

７．工学的応用

現在の社会においてニューラルネット応用が増えてきている。例えば、パターン認識、音声認識、画像認識、統計予測が挙げられる。コンピュータ技術の大きな柱として人工知能の応用が開発され、ニューラルネットや他の技術なども出てきた。本稿で分析したニューラルネットは人間や他の生物の脳にある神経回路網と比較すると、非常に単純である。それゆえ、パソコンの計算力が限定され、科学者は生物の脳をまだ完全に理解していないので、生物と同じように活動するニューラルネット技術はまだ作成されていない。しかし、テクノロジーは毎日進化しているので、将来の可能性はどうなるかのであろう。

以下の図は筆者が作成している単純な人柄ロボットで、完全に作り終わったら、パソコンに入っているニューラルネットでこのロボットを支配したいと思われる。現状は入力がないので、環境を知覚出来ない。それゆえ、何かの入力をインストールする必要がある。入力の例は圧力センサーや可視光線センサーなどが挙げられる。

図 7.1



８．実際の誤差逆電波法で学習するニューラルネット（ジャヴァ言語）

本稿の実験が使用したソフトは〈http://javagame.main.jp/index.php〉の「ニューラルネットによるパターン認識」というソフトから編集されている。

８．１ニューラルネットの主なクラス

```
/**
 * 多数隠れ層のニューラルネットのクラス ( 誤差逆伝播法 )
 * @author kenny
 * * (Original Source - http://javagame.main.jp/index.php)
 * 平成2008年3月13日
 */
public class NeuralNetwork {
	private Layer inputLayer;  // 入力層
	private Layer[] centerLayer;  // 中間層
	private Layer outputLayer;  // 出力層

	public void NeuralNetwork() {
		inputLayer = null;
		outputLayer = null;
	}

	/**
	 * ニューラルネットを初期化する
	 * @param numInputNodes  入力層のノード数
	 * @param numCL 中間層数
	 * @param numCenterNodes 中間層のノード数
	 * @param numOuputNodes  出力層のノード数
	 */
	public void init(int numInputNodes,int numCenterLayers, int numCenterNodes, int
```

```
numOutputNodes) {
		inputLayer = new Layer();
		centerLayer = new Layer[numCenterLayers];
		for(int i = 0; i < numCenterLayers; i++) {
		//	System.out.println("i = "+ i);
			centerLayer[i] = new Layer();
		}
		outputLayer = new Layer();
		// 入力層
		inputLayer.numNodes = numInputNodes;
		inputLayer.numChildNodes = numCenterNodes;
		inputLayer.numParentNodes = 0; // 入力層は親層がない
		inputLayer.init(numInputNodes, null, centerLayer[0]);
		inputLayer.setRandomWeights();
		// 中間層
		for(int i = 0; i < numCenterLayers; i++) {
			centerLayer[i].numNodes = numCenterNodes;
			if(i == 0) {//
				centerLayer[i].numParentNodes = numInputNodes;
				if(numCenterLayers == 1) {// もし中間層の最後の層だったら、出力層と
繋がる
					centerLayer[i].numChildNodes = numOutputNodes;
					centerLayer[i].init(numCenterNodes, inputLayer,
outputLayer); // 中間層数が一だから、親層＝入力層、子層＝出力層
				}
				else {
					centerLayer[i].numChildNodes = numCenterNodes;
					centerLayer[i].init(numCenterNodes, inputLayer,
centerLayer[i+1]);
				}
			}
			else { // 前層は入力層ではない
				centerLayer[i].numParentNodes = numCenterNodes;
				if(i == numCenterLayers - 1) {// もし中間層の最後の層だったら、出力
層と繋がる
					centerLayer[i].numChildNodes = numOutputNodes;
					centerLayer[i].init(numCenterNodes, centerLayer[i-1],
outputLayer);
				}
				else {
					centerLayer[i].numChildNodes = numCenterNodes;
					centerLayer[i].init(numCenterNodes, centerLayer[i-1],
centerLayer[i+1]);
				}
			}
			centerLayer[i].setRandomWeights();
		}
```

```
        // 出力層
        outputLayer.numNodes = numOutputNodes;
        outputLayer.numChildNodes = 0; // 出力層は子層がない
        outputLayer.numParentNodes = numCenterNodes;
        outputLayer.init(numOutputNodes, centerLayer[numCenterLayers-1], null);
        outputLayer.setRandomWeights();
}

/**
 *  入力層への一つの入力を設定する
 * @param i ノード番号
 * @param value 値
 */
public void setInput(int i, double value) {
        if (i >= 0 && i < inputLayer.numNodes) {
                inputLayer.neuronValues[i] = value;
        }
}

/**
 *  入力層への各入力を設定する
 * @param values 値
 */
public void setInputs( double[] values) {
        if(inputLayer.neuronValues.length == values.length) {
                inputLayer.neuronValues = values;
        }
}

/**
 * 出力層への一つの出力を得る
 * @param i ノード番号
 * @param value 値
 */
public double getOutput(int i) {
        if (i >= 0 && i < outputLayer.numNodes) {
                return outputLayer.neuronValues[i];
        }
        return Double.MAX_VALUE; // エラー
}

/**
 * 出力層への各出力を得る
 * @param values 値
 */
public double[] getOutput() {
        return outputLayer.neuronValues;
```

```
}

/**
 * 教師信号を設定する
 * @param i ノード数
 * @param value 教師信号の値
 */
public void setTeacherSignal(int i, double value) {
	if( i >= 0 && i < outputLayer.numNodes) {
		outputLayer.teacherSignals[i] = value;
	}
}

/**
 * 全ての教師信号を設定する
 * @param values 全ての教師信号の値
 */
public void setTeacherSignals(double[] values) {
	if (outputLayer.teacherSignals.length == values.length) {
		outputLayer.teacherSignals = values;
	}
}

/**
 * 入力層から出力層まで前向きを伝播する
 */
public void feedForward() {
	inputLayer.calculateNeuronValues();
	for(int i = 0; i < centerLayer.length; i++) {
		centerLayer[i].calculateNeuronValues();
	}
	outputLayer.calculateNeuronValues();
}

/**
 * 出力層から入力層まで逆向きに伝播する
 */
public void backPropagate() {
	outputLayer.calculateErrors();
	for(int i =  centerLayer.length - 1; i > -1 ; i--) {
		centerLayer[i].calculateErrors();
		centerLayer[i].adjustWeights();
	}
	inputLayer.adjustWeights();
}

/**
```

```
	 * 出力と教師信号の平均２乗誤差を計算する
	 * @return 平均２乗誤差
	 */
	public double calculateError() {
		double error = 0;
		for (int i = 0; i < outputLayer.numNodes; i++) {
			error += Math.pow(outputLayer.neuronValues[i] -
outputLayer.teacherSignals[i], 2);
		}
		error /=outputLayer.numNodes;
	 return error;
	}

	/**
	 * 学習率を設定する
	 * @param rate 学習率
	 */
	public void setLearningRate(double rate) {
		inputLayer.learningRate = rate;
		for(int i = 0; i < centerLayer.length; i++) {
			centerLayer[i].learningRate = rate;
		}
		outputLayer.learningRate = rate;
	}

}
```

８.２ニューラルネット層のクラス

```
import java.util.Random;
/**
 * ニューラルネット層のクラス
 * @author kenny
 * (Original Source - http://javagame.main.jp/index.php)
 * 平成2008年3月13日
 */
public class Layer {
	int numNodes;				// ノード数
	int numParentNodes;			// 親層のノード数
	int numChildNodes;			// 子層のノード数
	double[][] weights;			// この層と子層の結合荷重
	double[] neuronValues;		// ノードの活性値
	double[] teacherSignals;// 教師信号
	double[] errors;			// 誤差
	double[] biasValues; // バイアス値（閾値）
	double[] biasWeights;		// バイアスの重み
	double learningRate; // 学習率
```

```
Layer parentLayer;          // 親層
Layer childLayer;           // 子層

Random rand;                        // 層を初期化する時、重みをランダムに設定する

public Layer() {
        parentLayer = null;
        childLayer = null;
        rand = new Random();
}

/**
 * 層を初期化する
 * @param numNodes  この層のノード数
 * @param parent  親層
 * @param child  子層
 */
public void init(int numNodes, Layer parent, Layer child) {
        neuronValues = new double[numNodes];
        teacherSignals = new double[numNodes];
        errors = new double[numNodes];
        if (parent != null) {
                parentLayer = parent;
        }
        if (child != null) {
                childLayer = child;

                weights = new double[numNodes][numChildNodes];
                biasValues = new double[numChildNodes];
                biasWeights = new double[numChildNodes];
        } else {
                weights = null;
                biasValues = null;
                biasWeights = null;
        }
        // 重みを初期化する
        for (int i = 0; i < numNodes; i++) {
                neuronValues[i] = 0;
                teacherSignals[i] = 0;
                errors[i] = 0;
                if (child != null) {
                        for (int j = 0; j < numChildNodes; j++) {
                                weights[i][j] = 0;
                        }
                }
        }
```

```
		// バイアス値を初期化する
		if (child != null) {
			for (int i = 0; i < numChildNodes; i++) {
				biasValues[i] = -1;
				biasWeights[i] = 0;
			}
		}
	}

	/**
	 * 重みをランダムに設定する
	 */
	public void setRandomWeights() {
		rand.setSeed(System.currentTimeMillis());
		// 重みは-1.0〜1.0
		for (int i = 0; i < numNodes; i++) {
			for (int j = 0; j < numChildNodes; j++) {
				weights[i][j] = rand.nextInt(200) / 100.0 - 1;
			}
		}
		// バイアスも-1.0〜1.0
		for (int i = 0; i < numChildNodes; i++) {
			biasWeights[i] = rand.nextInt(200) / 100.0 - 1;
		}
	}

	/**
	 * 誤差を計算する
	 */
	public void calculateErrors() {
		if (childLayer == null) { // 出力層
			for (int i = 0; i < numNodes; i++) {
				errors[i] = (teacherSignals[i] - neuronValues[i]) * neuronValues[i]
* (1.0 - neuronValues[i]);
			}
		} else if (parentLayer == null) { // 入力層
			for (int i = 0; i < numNodes; i++) {
				errors[i] = 0.0;
			}
		} else { // 中間層
			for (int i = 0; i < numNodes; i++) {
				double sum = 0;
				for (int j = 0; j < numChildNodes; j++) {
					sum += childLayer.errors[j] * weights[i][j];
				}
				errors[i] = sum * neuronValues[i] * (1.0 - neuronValues[i]);
			}
```

```
		}
	}

	/**
	 * 誤差によると、結合荷重を調整する
	 */
	public void adjustWeights() {
		if (childLayer != null) {
			// 重みを調整する
			for (int i = 0; i < numNodes; i++) {
				for (int j = 0; j < numChildNodes; j++) {
					weights[i][j] += learningRate * childLayer.errors[j] *
neuronValues[i];
				}
			}
			// バイアスも調整する
			for (int i = 0; i < numChildNodes; i++) {
				biasWeights[i] += learningRate * childLayer.errors[i] *
biasValues[i];
			}
		}
	}

	/**
	 * この層の各ニューロンの活性値を計算する
	 */
	public void calculateNeuronValues() {
		double sum;
		if (parentLayer != null) {
			for (int j = 0; j < numNodes; j++) {
				sum = 0.0;
				// 親層の出力値と重みをかけて足す
				for (int i = 0; i < numParentNodes; i++) {
					sum += parentLayer.neuronValues[i] * parentLayer.weights[i]
[j];
				}
				// バイアス
				sum += parentLayer.biasValues[j] * parentLayer.biasWeights[j];
				// シグモイド関数を通す
				neuronValues[j] = sigmoid(sum);
			}
		}
	}

	/**
	 * シグモイド関数
	 */
```

```
	private double sigmoid(double x) {
		return 1.0 / (1 + Math.exp(-x));
	}
}
```

8.3 AND（論理積）の学習

```
/**

 * ANDの学習
 * @author kenneth cason
 * (Original Source - http://javagame.main.jp/index.php)
 * 平成2008年3月13日
 */
public class NeuralNetworkTest {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		NeuralNetwork nn = new NeuralNetwork();
		nn.init(2, 1, 2, 1); //  入力層のノード数、中間層数、中間層のノード数、出力層のノ
ード数
		nn.setLearningRate(0.5);

		// 訓練データの作成
		double[][] trainingSet = new double[4][3];
		// 訓練データ０
		trainingSet[0][0] = 0;		// 入力１
		trainingSet[0][1] = 0;		// 入力２
		trainingSet[0][2] = 0;		// 教師信号
		// 訓練データ2
		trainingSet[1][0] = 0;		// 入力１
		trainingSet[1][1] = 1;		// 入力２
		trainingSet[1][2] = 0;		// 教師信号
		// 訓練データ3
		trainingSet[2][0] = 1;		// 入力１
		trainingSet[2][1] = 0;		// 入力２
		trainingSet[2][2] = 0;		// 教師信号
		// 訓練データ4
		trainingSet[3][0] = 1;		// 入力１
		trainingSet[3][1] = 1;		// 入力２
		trainingSet[3][2] = 1;		// 教師信号

		// 訓練データを学習する
		double error = 1.0;
		int count = 0;
		while (error > 0.0001 ) { //&& count < 150000
```

```
                error = 0;
                count++;
                // 各訓練データを誤差が小さくなるまで学習する
                for(int i=0; i < 4; i++) {
                        // 入力層に値を設定する
                        nn.setInput(0, trainingSet[i][0]);
                        nn.setInput(1, trainingSet[i][1]);
                        // 教師信号を設定する
                        nn.setTeacherSignal(0, trainingSet[i][2]);
                        // 学習開始
                        nn.feedForward();
                        error += nn.calculateError();
                        nn.backPropagate();
                }
                error /= 4.0;
                System.out.println(count + "\t" + error);
        }
        // 学習完了
        nn.setInput(0, 0);    // 入力１
        nn.setInput(1, 0);    // 入力２
        nn.feedForward();     // 出力を計算する
        System.out.println("0 & 0 = "+nn.getOutput(0));
        nn.setInput(0, 0);    // 入力１
        nn.setInput(1, 1);    // 入力２
        nn.feedForward();     // 出力を計算する
        System.out.println("0 & 1 = "+nn.getOutput(0));
        nn.setInput(0, 1);    // 入力１
        nn.setInput(1, 0);    // 入力２
        nn.feedForward();     // 出力を計算する
        System.out.println("1 & 0 = "+nn.getOutput(0));
        nn.setInput(0, 1);    // 入力１
        nn.setInput(1, 1);    // 入力２
        nn.feedForward();     // 出力を計算する
        System.out.println("1 & 1 = "+nn.getOutput(0));
    }
}
```

９．参考文献

Wikipedia「バックプロパゲーション」＜http://ja.wikipedia.org/wiki/バックプロパゲーション＞（2008年1月17日アクセス）

Wikipedia「神経」＜http://ja.wikipedia.org/wiki/神経＞（2008年1月17日アクセス）

Wikipedia「List of animals by number of neurons」＜http://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons＞（2008年1月16日アクセス）

「ニューラルネットによるパターン認識」＜http://javagame.main.jp/index.php?%A5%CB%A5%E5%A1%BC%A5%E9%A5%EB%A5%CD%A5%C3%A5%C8%A4%CB%A4%E8%A4%EB%A5%D1%A5%BF%A1%BC%A5%F3%C7%A7%BC%B1＞（2008年1月16日アクセス）

小笠原 祐「誤差逆伝播法」＜
http://www.eb.waseda.ac.jp/murata/yu.ogasawara/openhouse/back_propagation.php＞
（2008年1月18日アクセス）

浅井潔（2006）「ニューラルネットの学習理論」
＜http://www.cbrc.jp/~asai/LECTURE/H16SeitaiJouhouRon/NN_learning.pdf＞
（2008年1月17日アクセス）