修士学位論文

ニューラルネットワークの開発支援システムに関する研究

東北大学大学院工学研究科
電気・通信工学専攻

宮野 靖弘

目 次

図目次

第1章

序論

1.1 本研究の背景

1940年代にコンピュータが出現してから半世紀以上の歳月が流れた．

この半世紀の間，情報処理システムは，高速性，および記憶容量においてその性能を著しく向上させ，現在，数値計算や演算処理においては，その正確性，緻密性においても人間の能力を凌駕するものになったといえる．この圧倒的なパワーの前に，チェスの世界チャンピオンが敗北を喫したことも記憶に新しい．しかし，人間の脳はこれらの情報処理システムをもってしても実現できない高度な能力を有している．例えば，文字を読取ったり，図形や物体を認識したり，言葉を聞き分けたりするようなことがそうである．我々はこれらのことを容易に成せるが，現在の情報処理システムで実現することは非常に難しい．これは演算速度やメモリの不足が原因であるとは考えにくい．

このような現在の情報処理システムの弱点を克服するためには，実際に人間の脳内でどのような情報処理が行われているのか，その原理を解明し，その原理をもとに新しい情報処理システムを構築することが最も自然な考え方といえる．

人間の脳における情報処理の仕組みを解明することは，以前は主に生理学者や心理学者によってなされてきたが，近年では工学的な知識をより多く得るため，工学者自身も脳の解明に力を注いでいる．工学者が脳の挙動から工学的な知識を得ようという場合，脳と同じ反応をする神経回路モデルを構成し，検証するという方法がよくとられる．このとき，生理学や心理学の実験によって解明されている事実をできるだけ忠実にモデルに取り入れ，解明されていない部分については大胆な仮説を立てモデルを作成する．こうして構成されたモデルを計算機シミュレーションや数学的解析によって調べ，検証を行う．この方法であれば，速やかにモデルを新しい情報処理システムに用いることができるし，こうして作成されたモデルは生理実験では困難な高次中枢の研究に対して威力を発揮している．[1]

このようにして提案されたモデルは数多く存在するが，その1つに，Fukushima-Miyakeらによる，生物における視覚神経系を基礎に構成されたモデルであるネオコグニトロン(Neocognitron) [1, 2]がある．この神経回路モデルは視覚情報の統合，そして，パターン認

識機構をモデル化したものであり，脳の大きな特徴である学習と自己組織化の能力を備えている．計算機によるパターン認識シミュレーションでは，ノイズや変形，位置ずれに優れた頑健性を示しており，これからも発展が望まれる神経回路モデルであると考えられる．

近年，ネオコグニトロン型神経回路モデルは，回転したパターンの認識 [3]，バインディング問題のシミュレーション [4] など，視覚系の高度な情報処理のみならず，連想想起のモデルなどで使用されるケース [5] もあり，様々な工夫が施され，高機能化が図られている．しかし，この高機能化のため，その構造が初期のモデルより複雑化しており，また，研究の目的毎に多種のネオコグニトロン型神経回路モデルが存在する．よって，研究者が計算機によるシミュレーションを行う際には，一人一人が1からネオコグニトロン型神経回路モデルを設計し，しかも構造が複雑な場合には非常に長い時間をプログラミングに費やさなければならないのが現状となっている．

1.2 本研究の目的

本研究は，視覚情報処理神経回路モデルのうち，特にネオコグニトロン型神経回路モデルに注目し，これを計算機上で動作させる際に研究者が速やかにモデルをプログラミングできるようなシステムを作成することを目的とする．そのため，神経回路モデルを作成する際に利用できるプログラムライブラリを作成した．

このライブラリは，ネオコグニトロン型神経回路モデルの各構成要素をクラス化したものによって構成され，特にネオコグニトロン型モデルの作成に適したものとなっているが，ネオコグニトロンの階層的な構造をライブラリの構造に取り入れたことにより，その他のモデルにも用いることが可能となっている．

このライブラリを利用することにより，ネオコグニトロン型神経回路モデルの基礎的，共通的な部分のプログラミングについては研究者一人一人が負担することなく，モデルの構築をより容易に行うことができる．

1.3 本論文の構成

本論文の構成は以下の通りである．

第1章 は序論であり，本研究の背景，目的について述べた．

第2章 では，ネオコグニトロン型神経回路モデルの構成，動作，学習について説明する．

第3章 では，ライブラリの設計方針，構成について説明する．

第4章 では，ライブラリの評価について述べ，また，応用例を示す．

第5章 は結論であり，本論文のまとめ，および今後の課題について述べる．

第2章

ネオコグニトロン

この章では，ニューラルネットモデルを作成する際に最も基本的な部分である神経細胞の説明から，その神経細胞を組み合わせて作成される神経回路モデル，ネオコグニトロンについて，その背景にどのような仮説や生理学的実験結果があるのかを述べる．

2.1 神経細胞

脳は巨大な数の神経細胞が結合してできた大規模システムであるといわれており，人間の場合，脳の神経細胞の数は 10^{10} 個以上といわれている．ここでは，脳の構成要素である神経細胞について，近年の生理学における研究から明らかになっていることについて述べる．

2.1.1 神経細胞の構造

脳は多数の神経細胞 (neuron, ニューロン) がつながりあった神経回路 (neural network, ニューラルネットワーク) からなる．大脳や小脳にはこの神経細胞が 1 mm^3 中に数万個も詰め込まれている．神経細胞の形状は様々に異なっているが，基本的には細胞体 (cell body) とそれから出る多くの突起からなっている．細胞体の大きさは，およそ，数 μm ～ $100\mu m$ 程度である．細胞体から出ている突起は，軸索 (axon) とよばれる 1 本の細くて長い繊維と，樹状突起 (dendrite) とよばれる木の枝のように広がっている比較的太くて短い多数の突起とに分けられる．軸索の役割りは細胞体の反応出力を他の神経細胞に伝送することで，その先端は枝分れしており，終端部は他の神経細胞の樹状突起または細胞体に接触している．この接触点をシナプス (synapse) という．神経回路の情報のやりとりは，すべてこのシナプスを介して行われ，その伝達方向は一方行である．

神経細胞は膜電位につつまれており，膜電位内部は外部より約 $-70mV$ の電位に保たれている．これを静止膜電位という．やがて他の神経細胞からパルスを受けとり，膜電位が

あるしきい値 (約 $-55mV$) を越えたとき，振幅 $100mV$ ，持続時間 $1m$ sec 弱程度の正のパルス電位を発生する．これを興奮，または発火という．

発火した後は再びもとの静止電位にもどるが，これまでの間にいくら強い刺激を与えても再び発火しない期間が存在し，これを絶対不応期という．絶対不応期の後から静止電位に戻る前の間，神経細胞は発火しにくい状態にあり，これを相対不応期という． [6]

2.1.2 シナプス

シナプスでは，軸索の終端部が他の神経細胞の細胞体や樹状突起に 10〜15 nm 程度の間隙 (synaptic cleft, シナプス間隙) をはさんで接している．細胞体に発生したパルスが軸索の終端に達すると，アセチルコリンなどの化学伝達物質がシナプス間隙に向かって放出され，シナプス後部 (情報を受けとる側) に正，または負の電位を発生させる．このようにしてシナプス後部に発生した電位を PSP(postsynaptic potential) という．ここで，正の PSP を発生させるシナプスは，他の神経細胞の発火を促すことになるので，興奮性シナプス (excitatory synapse) といい，このシナプスによって生じた正の PSP を EPSP(excitatory PSP, 興奮性シナプス後電位) とよぶ．逆に負の PSP を発生させるシナプスは他の神経細胞の発火を抑えることになるので，抑制性シナプス (inhibitory synapse) といい，このシナプスによって生じた PSP を IPSP(inhibitory PSP, 抑制性シナプス後電位) とよぶ． [1]

2.2 神経細胞のモデル

神経細胞のモデルを作成する場合に，神経細胞のどのような性質に着目するかによって種々の異なったモデルが考えられる．神経細胞そのものの生理作用を調べるのを目的とする場合はには，神経細胞の細部を忠実にモデル化することが必要であるが，これに対し，多数の神経細胞が組み合わされている神経回路の性質を調べることを目的とするならば，神経細胞単体のモデルとしてはあまり複雑なものを用いず，単純化したモデルを採用するのが全体の働きを把握しやすいといえる．

ここでは，後者のような目的で作成された神経細胞モデルのうちで，一般的なものについて述べる．

2.2.1 モデルとしての神経細胞

神経細胞は，多入力 1 出力の情報処理素子といえる．N 個の神経細胞から入力信号を受け取る神経細胞を考え，これらの信号の強さを $u_1, u_2, \cdots, u_n$ とする．また，興奮のしきい値を θ ，出力信号を v とする．第 n 番目の軸索に単位の強さの信号が流れたときに，この影響を受けて変化する膜電位の量を c_n とする．c_n はシナプスの結合効率を表わす量であり，シナプス荷重，結合重み，または単に重み，とよばれる．通常，興奮性シナプスに対しては， $c_n > 0$ ，抑制性シナプスに対しては，$c_n < 0$ となる．

神経細胞の入出力関係を表すモデルを作る前に，神経細胞の特徴を列挙する．

1. 空間的加算

膜電位の変化は，多数の入力信号の重ね合わせで決まる．n 番目の入力に強さ u_n の信号が来るとき，膜電位は $c_n u_n$ だけ変化する．したがって，全体の膜電位は，入力信号の重みつき線形和

$$\sum_{n=1}^{N} c_n u_n \tag{2.1}$$

に関して変化する．

2. 時間的加算
入力信号の影響は時間的にしばらくの間持続し，あとから来る入力信号の影響と重なり合う．n 番目の入力に来る単位の強さの入力が t' 時間後の膜電位に及ぼす影響を $c_n(t')$ とする．すると，時間 t の膜電位の変化分 $u(t)$ は，時間和・空間和を考慮に入れると

$$\sum_{n=1}^{N} \int_0^1 c_n(t-t')u_n(t')dt' \tag{2.2}$$

に関係することになる．ただし，$u_n(t')$ は時間 t' における入力信号の強さである．この式は，シナプス—樹状突起が一種の線型フィルタの役割を果たすことを示している．

3. しきい値作用
出力 v は膜電位がしきい値 θ に達するまでは発生しない．このことは，入出力関係が非線型になっていることを示している．

4. 不応期
しきい値 θ は定数ではなく，神経の興奮によって変化する．絶対不応期は，θ の値が ∞ に上がった期間とみなせる．

5. 疲労
しきい値 θ は，興奮が続くにつれて，さらに長期的な変化をたどる．

6. 可塑性
学習は，結合の荷重 c_n の変化によってもたらされると考えられる．この考えは多くの自己組織神経回路モデルで採用されている．[6, 7]

2.2.2 しきい素子型モデル

しきい素子型モデル

神経細胞が発火して発生するパルスの振幅や時間幅はほぼ一定である．そこで，神経細胞は興奮している状態 (発火している瞬間) と静止している状態の二つの状態しか持たないとみなし，出力として 1 または 0 のいずれかの値をとるしきい素子 (threshold element) を，神経回路のモデルと考えることができる．

この素子は 図2.1 のように多入力 1 出力で，各入力端子はそれぞれ 1 個のシナプスに対応する．各入力端子には，シナプス結合の強さ，結合係数 (シナプス荷重) を割り当てる．全入力の加重和が細胞体内部に発生した PSP に対応すると考え，PSP が一定のしきい値 θ を越えると素子は出力 1 を出し，PSP がしきい値以下のときには出力は 0 とする．

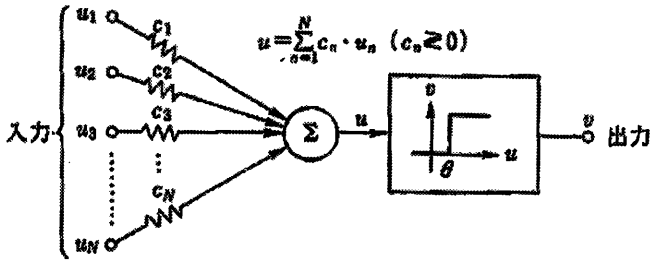


図 2.1: しきい素子

すなわち，n 番目の入力端子 (シナプス) に加えられる入力信号を u_n とし，その入力端子の結合係数 (シナプス荷重) を c_n とすると，この素子の出力 v は

$$v = 1\left[\sum_{n=1}^{N} c_n \cdot u_n - \theta\right] \tag{2.3}$$

で与えられる．ここに，θ はしきい値を定める定数であり，1[ ] は単位段階関数で，次式で定義される．

$$1[x] = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.4}$$

なお，この場合，u_n を他の神経細胞の出力と考えれば，u_n もやはり 1 または 0 の 2 値をとる変数である．

このモデルでは，多数のシナプスからの入力の空間的加重特性にだけ注目しているが，時間特性を考慮たモデルも提唱されている．

MuCulloch-Pitts のモデルは，時間特性を考慮したシンプルなしきい素子型モデルであり，上記のしきい素子に遅延時間を加味したものである．神経回路を構成するすべての神経細胞は時刻 τ ごとに同期して状態を変え，出力 1 または 0 を出すと考える．いま，時刻 t における n 番目の神経細胞の出力を $v_n(t)$ とすると，次の関数が成立する．

$$v_n(t+\tau) = 1\left[\sum_{m=1}^{N} c_{nm} \cdot u_m(t) - \theta_n\right] \tag{2.5}$$

ここに，c_{nm} は n 番目の神経細胞へのシナプス結合の強さを，θ_n は n 番目の神経細胞のしきい値を表す．

Caianiello は，さらに不応期を考慮したモデルを考えている．すなわち，n 番目の神経細胞の出力は，

$$v_n(t+\tau) = 1\left[\sum_{m=1}^{N}\sum_{r=0}^{\infty} c_{nm}^{(r)} \cdot u_m(t-r\tau) - \theta_n\right] \tag{2.6}$$

で与えられるものとする．この式で，$n = m$ となる結合係数 $c_{nn}^{(r)}$ によって不応期を導入する．つまり，絶対不応期の持続時間を表す正の定数 T_1 と相対不応期の持続時間を表す正

の定数 T_2，および適当な正の単調減少関数 $f(\ \ )$ を用いて，$c_{nn}^{(r)}$ を次のように定義する．

$$c_{nn}^{(r)} = \begin{cases} -L & (0 \le r < T_n) \\ -f(r) & (T_1 \le r < T_1 + T_2) \\ 0 & (r \ge T_1 + T_2) \end{cases} \tag{2.7}$$

ここに，L は大きな正の定数である．

また，PSP の加算に時間がかかることを考慮した次のようなモデルが用いられることもある．すなわち，n 番目の神経細胞の時刻 t における PSP を $u_n(t)$，その神経細胞の出力を $v_n(t)$ とするとき，次の微分方程式が成立すると考える．

$$v_n(t) = 1[u_n(t) - \theta_n] \tag{2.8}$$

$$\frac{du_n(t)}{dt} = -\alpha \cdot u_n(t) + \sum_{m=1}^{M} c_{nm} \cdot u_m(t - \tau_{nm}) \tag{2.9}$$

ここに，τ_{nm} は m 番目の神経細胞が n 番目の神経細胞に伝えられるまでの時間遅れで，軸索をパルスが伝播する時間やシナプス遅延等を含めた遅延時間を表す．α は $0 < \alpha < 1$ を満足する定数であり，c_{nm} はシナプスの強度の強さを表す． [1]

2.2.3 アナログ型モデル

神経細胞はパルス出力を出すが，少なくとも抹消系に近い箇所では，神経パルス1個1個が意味を持つというよりは，むしろパルス密度の形でアナログ処理的な情報を伝えていると考えられる．そこで次に，パルス密度に比例したアナログ値を入出力とした素子について述べる．

アナログしきい素子

神経細胞の出力がパルス密度に比例した値であると考えると，負になるのは不合理である．そこで，出力が非負 (正または 0) のアナログ値をとるアナログしきい素子が考えられた．

アナログしきい素子とは，あるしきい値を定め，前述のしきい素子と同様に全シナプスからの入力の加重和を求め，その加重和がしきい値以上のときには加重和そのものを出力とするが，加重和がしきい値以下であった場合には出力を 0 とするものである．最も簡単なものは，このしきい値を 0 に選んだ非直線特性を持つものである．

n 番目の入力端子 (シナプス) からの入力を u_n とし，その入力端子の結合係数を c_n とすると，この素子の出力 v は，

$$v = \varphi\left[\sum_{n=1}^{N} c_n \cdot u_n\right] \tag{2.10}$$

で与えられる．ここに，$\varphi[\ \ ]$ は折線型 (半波整流形) の関数

$$\varphi[x] = \begin{cases} x & (x \ge 0) \\ 0 & (x < 0) \end{cases} \tag{2.11}$$

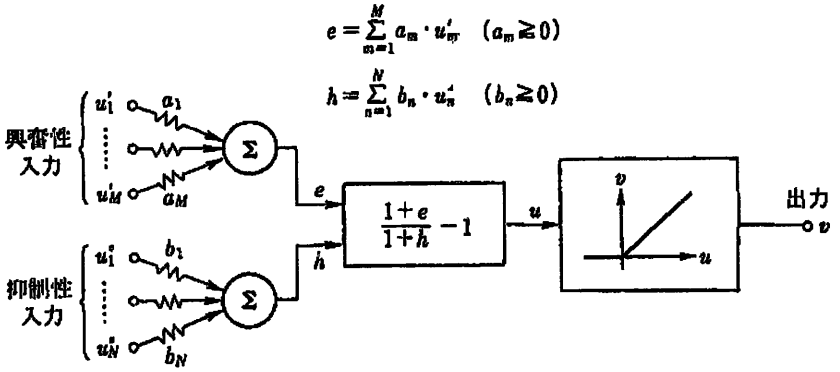


図 2.2: 分流抑制機構を持つ素子

を表す.

このほか，しきい値を 0 以外の値に選んだ素子，すなわち入出力特性が

$$v = \varphi\left[\sum_{n=1}^{N} c_n \cdot u_n - \theta\right] \tag{2.12}$$

で与えられる素子を考えることもできる.

また，非線型関数 $\varphi[\ ]$ の変わりに飽和特性を持った S 字型の関数を用いたモデルもある．たとえば，双曲線関数 tanh を用いたモデルは，入力のあらゆる値に対して微分可能性が要求されるような神経回路モデル (バックプロパゲーションを用いた教師なし学習のモデルなど) にも採用されている.

アナログしきい素子はこのように非常に単純化されてはいるが，情報処理において本質的な役割を果たしている空間的加重和としきい特性の両者を含んでいるので，複雑な神経回路のモデル化を行う場合に都合がよい． [1]

2.2.4 分流型抑制機構をもつ素子

以上の素子では，抑制性のシナプスからの入力は EPSP に対して引算的に作用すると考えているが，抑制が EPSP の効果を分流 (shunt) するように働くとしたモデルも考えられている.

福島らは， 図 2.2 のようなモデルを提案した．興奮性入力を $u'_1, \cdots, u'_M$ とし，抑制入力を $u''_1, \cdots, u''_N$ とするとき，素子の出力 v は,

$$v = \varphi\left[\frac{1 + \sum_{n=1}^{M} a_m \cdot u'_m}{1 + \sum_{n=1}^{N} b_m \cdot u''_n}\right] \tag{2.13}$$

で与えられると仮定している．入出力はやはり非負のアナログ値で，$\varphi[\ ]$ は前述の 式 (2.11) で定義される関数である．また，a_m と b_m はそれぞれ，興奮性および抑制性のシナプス結合の強度を表す正の定数である．このモデルでは，抑制性入力は EPSP を負の一定電位 (IPSP の平衡電位) に向かって分流させるように働くと考えている．

いま，この素子に対する興奮性入力の総和を e，抑制性入力の総和を h と表す．すなわち，

$$e = 1 + \sum_{m=1}^{M} a_m \cdot u'_m \tag{2.14}$$

$$h = 1 + \sum_{n=1}^{N} b_n \cdot u''_n \tag{2.15}$$

このよな記号を用いると，この素子の出力は

$$v = \varphi\left[\frac{1+e}{1+h} - 1\right] = \varphi\left[\frac{e-h}{1+h}\right] \tag{2.16}$$

と表すことができる．したがって，抑制入力 h が小さいときには近似的に $v \fallingdotseq \varphi[e-1]$ になり，式 (2.10) のアナログしきい素子の入出力特性に一致する．しかし，入力が大きくなると，$v \fallingdotseq \varphi[e/h - 1]$ となり，出力は興奮性入力 e と抑制性入力 h との差ではなく比で定まるようになる．

ここで，興奮性入力 e と抑制性入力 h とが比例して増加する場合の入力特性は，$e = \varepsilon x$，$h = \eta x$ と書くと，$\varepsilon > \eta$ の場合には，式 (2.16) は，

$$v = \frac{(\varepsilon - \eta)x}{1 + \eta x} = \frac{\varepsilon - \eta}{2\eta}\left\{1 + \tanh(\frac{1}{2}\log \eta x)\right\} \tag{2.17}$$

と変形できる．これは，Weber-Fecher 則で表される対数特性に tanh で表される S 字特性を加味した入出力特性[1] になり，生理学や心理学での感覚系の入出力の非直線特性を近似する実験式として広く用いられているものにも一致する．したがってこの素子は，神経系における非直線特性の本質をとらえた神経細胞のモデルといえる． [1]

2.3 視覚情報処理

2.3.1 視覚と大脳

視覚野は左右の大脳半球の後部 (後頭野) ある．Broadmann の脳地図 図 2.3 の 17 野 (area17; striate area; striate cortex) が第 1 次視覚野 (V1=visual area I) に対応する．狭義には 17 野だけを視覚野とよび，その前方に位置する 18 野 (area 18) および 19 野 (area 19) を視覚前野 (prestriate visual cortex) とよぶ．

[1] Weber-Fecher 則とは，ある刺激によって生じる感覚の強さが，その刺激の強度 I の対数に比例するという法則である．この法則は，刺激の強度を増加したときの弁別しきい値 ΔI とそのときの刺激強度 I との間には $\Delta I/I =$ 一定，という関係が成立するという実験結果 (Weber の法則) をもとにして導かれたものである．しかし，種々の感覚に対して心理測定を行ってみると，通常，$\Delta I/I =$ 一定 となるのは，刺激強度がある範囲内にあるときだけで，刺激強度が非常に小さいときや大きいときには ΔI の値は一般に増大する．このことから推測すると，$\log I$ を横軸に感覚量を縦軸にとって描いたフラフが直線になるのはその中域部のみであって，両端部では S 字型の飽和特性を示すことが推定される． [1]

最近の研究により，視覚前野は，機能的に異なった多数の領域に分けられていることがわかってきた．現在は，V2(第2次視覚野； viaual area II)，V3, V3A, V4, MT (middle temporal area), MST(medial superior temporal area) などと名づけられる領域に分けられている．なお，視覚に関連した領域は，後頭葉だけでなく，下部側頭葉 (inferotemporal cortex; 側頭連合野の下部に位置する) の後半部の PIT 野 (posterior inferotemporal area) や，前半部の AIT 野 (anterior inferotemporal area)，頭頂葉野 (頭頂連合野) の 7a 野，さらには前頭葉などにも及ぶことがわかってきている．PIT 野と AIT 野とをまとめて IT 野 (inferior temporal area) とよぶこともある． [1, 8]

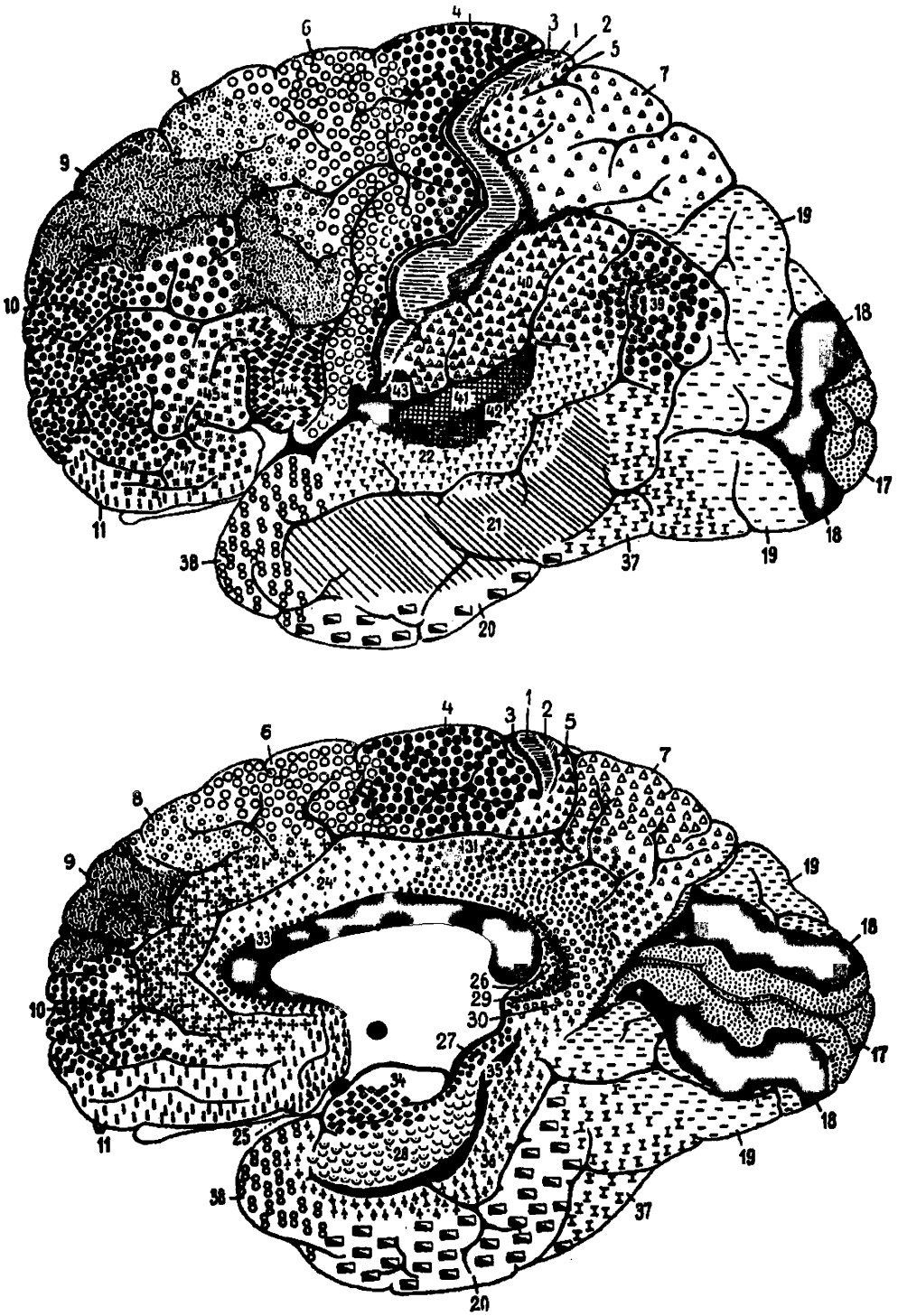


図 2.3: Brodmann の脳地図
上図はヒトの大脳の左半球を外側から見た図，下図は右半球を内側から見た図．
図の左側が前，右側が後．

2.3.2　Hubel-Wiesel の階層仮説

大脳の中で視覚情報を最初に受け取って処理しているのが，視覚野である．視覚野には，特定の傾きの直線成分や輪郭に反応する細胞を始めとして，網膜上に投影された刺激パターンに含まれる種々の特徴に反応する細胞が存在することが知られている．

Hubel と Wiesel は 1960 年代に，ネコやサルの視覚野 (V1 野と V2 野) の神経細胞の受容野の性質を調べ，これらの細胞をその受容野の性質に基づいて，単純型細胞 (simple cell) 複雑型細胞 (complex cell)，超複雑型細胞 (hypercomplex cell) などに分類し，これらの細胞相互間には，網膜→外側膝状体→単純型細胞→複雑型細胞→超複雑型細胞，という階層的な構造が存在するという階層仮説を提唱した．[1]

以下に，本研究と関係深い単純型細胞と複雑型細胞について述べる．

単純型細胞

単純型細胞の受容野の例を 図 2.4 に示す．

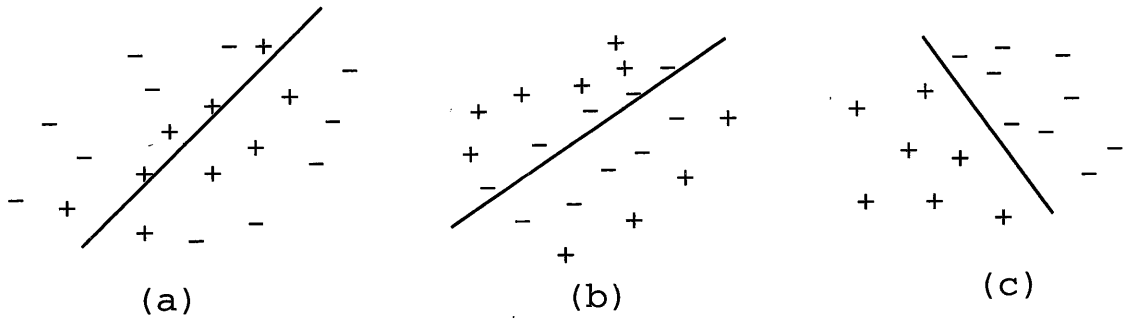


図 2.4: 種々の単純型細胞の受容野：+ オン反応，- オフ反応

図は網膜上のいろいろな場所を光のスポットで刺激したとき，今観測している 1 個の神経細胞がどのように反応するかを示したもので，図中の＋印の位置の刺激に対してはオン反応を示し，－印の位置に刺激に対してはオフ反応を示すことを表している．単純型細胞の受容野はこのようにいろいろな形のものが存在するが，共通していえることは，オン領域とオフ領域とが，円形ではなく，直線で分割されていることである．

これらの細胞では，受容野内の興奮性のオン領域に刺激光が与えられると発火の頻度 (出力パルスの頻度) は増加し，抑制性のオフ領域に刺激光が与えられると発火は抑制される．刺激光の面積を増すとこれらの効果は増大し，興奮性領域，抑制性領域のそれぞれの中で，興奮，抑制の効果が加算されていく．しかし，両領域に同時に刺激を与えると両者の効果は互いに打ち消し合い，受容野全体を覆うような一様光の刺激に対してはほとんど反応しない．したがって，刺激のパターンの形や位置が受容野の興奮性領域 (＋印) にちょうど一致している場合に強い反応を示す．たとえば，図 2.4 の (a) のような受容野を持った細胞の場合には，＋印の部分に一致するような直線状の刺激光に最も強く反応を示す．この直線状の刺激光の方位 (直線の傾きや；orientation) や位置がずれると，－印の部分にも刺激光が与えられることになり，反応は抑えられてしまう．そこで，このような細胞は，特定の位置に存在する特定の方位を持った明るい (白い) 直線の検出に役立っているのであろう

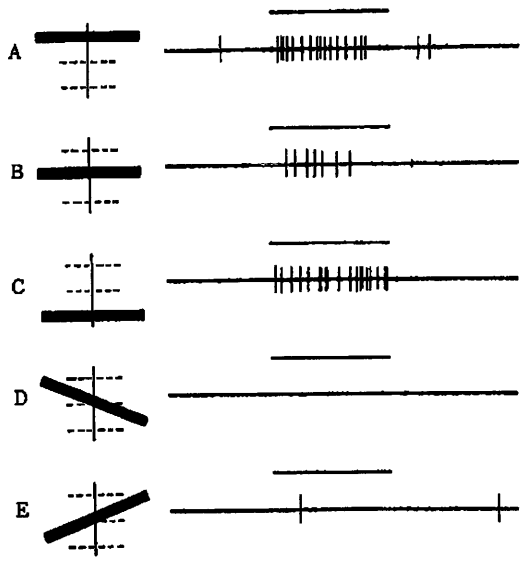


図 2.5: 複雑型細胞の反応の一例

と推定される．また，同図の (b) のような受容野は黒い直線の検出に，同図の (c) のような受容野は特定の方位のエッジ (白と黒の境界) の検出に役立っていると考えられる．つまり，単純型細胞は，明暗の境界や線が視界内のどの位置にあり，どのような方位を持っているかの検出を行なっているものと考えられる． [1]

複雑型細胞

単純型細胞では，刺激として与える直線やエッジの位置が少しでもずれると出力は抑制されてしまうが，これに対して複雑型細胞では，直線やエッジが受容野の内部にありさえすれば，その位置にはあまり影響されずに反応しつづける．しかし，刺激パターンの方位や運動方向には敏感に影響を受け，方位や運動方向がずれると，たとえ受容野内に刺激が与えられていても反応しない． 図 2.5 に複雑型細胞の反応の一例を示す．図の左側は刺激パターンを示し，右側はその刺激に対する細胞の反応出力 (横軸は時間) を示す．出力パルスの上に引いた横線は，刺激パターンが提示されている時間を示す．この細胞は直線刺激に対して強い反応を示すが，細胞によってはエッジに対して強く反応するものもある．

なお，複雑型細胞も，単純型細胞と同様に，一般に静止した刺激よりも運動刺激に対してとくに強い反応を示し，運動方向や速度に対しても最適条件が存在する． 図 2.6 は，運動刺激に対する複雑型細胞の反応の一例である．運動方向や速度に対する反応の対称性 (前進方向と後進方向の動きに対する反応の違い) は細胞によって違い，対称なものと非対称なものの両者がある．

複雑型細胞は，光のスポットの刺激にはあまり反応せず，また，反応したとしてもオン—オフ反応 (光をつけた瞬間にも消した瞬間にも反応出力が増加する反応様式) を示す．したがって単純型細胞とは異なって，受容野の内部をオン領域とオフ領域とに分割すること

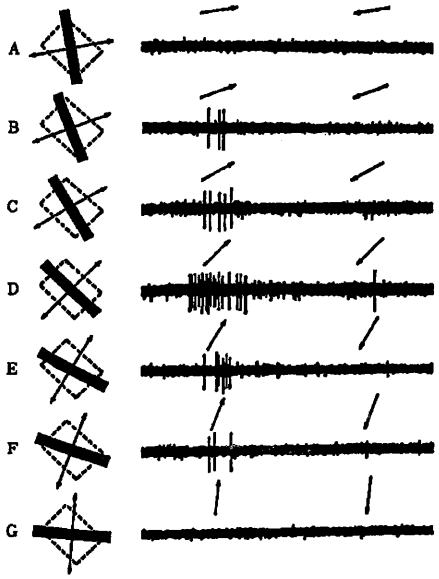


図 2.6: 運動刺激に対する複雑型細胞の反応

ができず，たとえば 図2.4のような形式で受容野を表示することはできない．

Hubel と Wiesel は，複雑型細胞は，同一の最適方位を持つ複数個の単純型細胞から興奮性シナプス結合を受けとっており，これらの単純型細胞のうちのいずれか1個でも反応すれば複雑型細胞も発火するようになっていると考えた． 図2.7は，直線刺激に強い反応を示す複雑型細胞を例にとって，その神経回路を図示している．この複雑型細胞に興奮結合をしている単純型細胞は，いずれも同一方位の直線刺激に反応するが，その受容野の位置は互いに少しずつずれている．したがって，刺激パターンの位置が多少変わってもその方位さえ適当であれば，これらの単純型細胞の中の少くとも1個は反応出力を出し，その出力を受けて複雑細胞も発火することになる． [1]

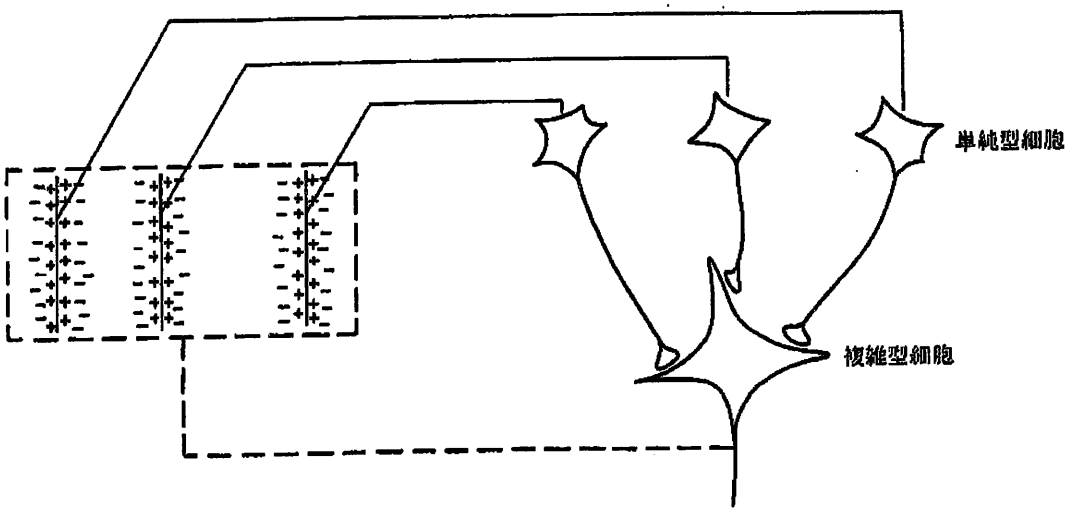


図 2.7: 複雑型細胞のシナプス結合モデル

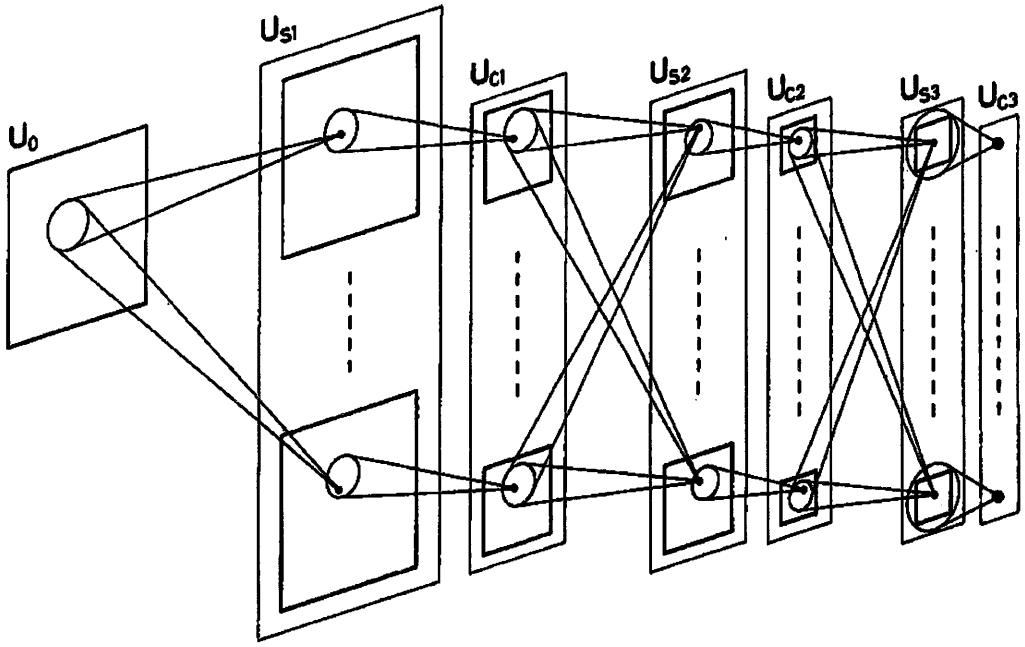


図 2.8: ネオコグニトロンの各層の細胞間の結合状態を示す概念図

2.4 ネオコグニトロンの概要

ネオコグニトロン (Neocognitron) [1, 2] は，前章で述べた Hubel-Wiesel の階層仮説，単純型細胞，複雑型細胞など，生理学的にわかっている事実をもとにつくられたモデルである．

ネオコグニトロンは，図 2.8 に示すように，階層構造をもつ多層の神経回路である．入力層 U_0 には，光受容細胞 (光電変換素子) が平面上にならんでいる．入力パターンが工学的に投影されると，各光受容細胞は受光した光りの強さに応じた出力を出す．入力層 U_0 の後ろには，S 細胞と名付けた細胞の層 U_s と，C 細胞と名付けられた細胞の層 U_c とが交互に並んでいる．S 細胞は，神経生理学でいう単純型細胞 (simple cell) に似た反応特性を示す細胞であり，C 細胞は複雑型細胞 (complex cell) に似た反応特性を示す細胞である．各細胞層内には多数の細胞が並んでおり，それらの各細胞は図 2.8 に示すように，自分よりも 1 段前の層の細胞のうちで，ある小領域に存在する一群の細胞だけから入力を受けている．階層の最上位段の C 細胞がネオコグニトロンの最終的な認識結果を示す認識細胞である．学習を終わったネオコグニトロンの最上位段 (認識細胞層) では，刺激パターンのカテゴリーに対応する認識細胞が 1 個だけ出力を出す．

回路内の細胞の入力結合には，初期状態においてすでに結合が完成している固定結合と，外部から与えられる刺激の状態に応じて結合の強さが変化していく可変結合の両者がある．

S 細胞は特徴抽出機能をもつ細胞で，その入力結合は可塑性を持ち，学習時には最大値検出型仮説[2] に従って強化されていく．学習が完了したあとの S 細胞は，その前段の一群

[2]ある小さな領域 (競合領域とよぶ) 内に存在する一群の細胞のうちで，最大出力を出した細胞 1 個だけが，入力シナプスの強化を受け，しかも，このとき，この細胞に結合している多数の入力シナプスが全部強化さ

のC細胞 (つまり，S細胞の結合領域 (又は結合範囲) に含まれるC細胞) の反応の空間パターンが，ある特定のパターンを示した場合のみ出力を出すようになる．つまり，各S細胞は，入力パターンの局所的な特徴の1つに選択的な反応を示すようになる．

このような一群の細胞は，各細胞内で2次元平面状に集まって形成されるので，これを“細胞面” とよぶことにする．一つの細胞層内にはこのような細胞面が多数存在する．図2.8において，太線で囲んだ四角形が個々の細胞面を表している．したがって学習終了後には，同一細胞面内のS細胞は，いずれも同一形状の特徴に反応する (つまり同一構造の受容野を持つ) ようになるが，その特徴を抽出してくる入力層上の位置 (つまり受容野の位置) は細胞ごとに異なっている．

図2.8では，見やすさの点から一つの細胞面内の1個の細胞に至る結合だけしか描いていないが，実際には一つの細胞面内に含まれる細胞は，どの細胞をとってみても，まったく同一の空間分布の入力結合を持っている．一つの細胞面内の細胞相互間の相違は，その結合する相手の細胞がちょうど面内の細胞の位置ずれに相当する距離だけ平行移動しているだけである．

S細胞の入力結合の強度が学習によって変化するのに対し，C細胞は学習によって変化しない固定した入力結合を持っている．つまり，各C細胞は，特定の一つの細胞面内にある一群のS細胞 (特徴抽出細胞) から興奮性の固定結合を受け取っていて，その入力側のS細胞が1個でも出力を出せばC細胞も出力を出すようになっている．したがって，C細胞は入力側の個々のS細胞に比して特徴提示位置のずれにあまり影響を受けなくなっている．

入力パターンの情報はこのように特徴抽出細胞層であるS細胞層と，C細胞層を何段も経て最上位段のC細胞層に伝えられる．多層回路各段での特徴抽出と統合の過程で，特徴相互の位置ずれが少しずつ許容されていくので，最上位段のC細胞は，入力パターンの位置ずれをほとんど受けずに，形の違いに対してのみ反応する．

2.5 ネオコグニトロンの構造

2.5.1 ネオコグニトロンの構造と定式化

はじめに，l 段目のS細胞の層を U_{sl} ，C細胞の層を U_{cl} と記す．各細胞層は，細胞面に分かれている．この細胞面の数を K_{sl} または，K_{cl} とする．各細胞面には，S細胞あるいはC細胞がそれぞれ平面状にならんでおり，U_{sl} 層の k 番目の細胞面に含まれる1個のS細胞の出力を $u_{sl}(\boldsymbol{n}, k)$ と表す．$\boldsymbol{n} = (n_x, n_y)$ は，その細胞の受容野の中心位置を示す2次元座標である．S細胞は， 図2.9のように前段のC細胞から結合を受けている．S細胞には，補助的な働きをする抑制性のV細胞が含まれている．このV細胞の出力を $u_{vl}(\boldsymbol{n}, k)$ と表す．V細胞は，1つのS細胞層に対し，そのS細胞層内の1つの細胞面と同じ数存在

れるのではなく，その瞬間にパルス入力が到来していた入力シナプスだけが強化される，という仮説． [1]

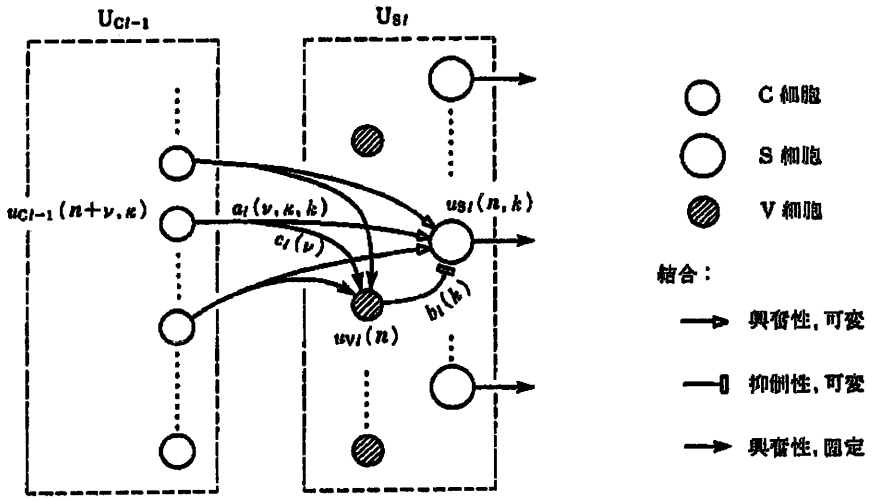


図 2.9: S 細胞への結合

し，1つの V 細胞は，K_{sl} の S 細胞に抑制信号を送っている．S 細胞の出力の式は，

$$u_{sl}(\boldsymbol{n},k) = r_l \cdot \varphi\left[\frac{1+\sum_{\kappa=1}^{K_{cl}-1}\sum_{\nu\in A_l} a_l(\boldsymbol{\nu},\kappa,k)\cdot u_{cl-1}(\boldsymbol{n}+\boldsymbol{\nu},k)}{1+\dfrac{r_l}{1+r_l}\cdot b_l(k)\cdot u_{vl}(\boldsymbol{n})} - 1\right] \tag{2.18}$$

となる．ただし，$\varphi[\ ]$ は，式 (2.11) で示した折線型の非線型関数である．

$a_l(\boldsymbol{\nu},\kappa,k)$ は，その前段の κ 番目の細胞面内の C 細胞で，位置が ν だけずれたところにあるもの，つまり，$u_{cl-1}(\boldsymbol{n}+\boldsymbol{\nu},k)$ からの興奮性可変結合の強度である．A_l は，1個の S 細胞が入力を受け取る範囲の広さ (結合可能領域) を表す．$b_l(k)$ は，同じ層の V 細胞 $u_{vl}(\boldsymbol{n})$ からの抑制性可変結合の強度であり，抑制は S 細胞に対して分流的に働いている．

r_l は，抑制入力を制御する正の定数であり，この値が大きくなると，S 細胞が特定の一つの特徴にだけ反応するようになる．

式 (2.18) に抑制性の入力信号を送っている V 細胞 $u_{vl}(\boldsymbol{n})$ は，$u_{sl}(\boldsymbol{n},k)$ が受け取るのと同じ細胞群から信号を受け取り，その出力は

$$u_{vl}(\boldsymbol{n}) = \sqrt{\sum_{\kappa=1}^{K_{cl}-1}\sum_{\nu\in A_l} c_l(\nu)\cdot\{u_{cl-1}(\boldsymbol{n}+\boldsymbol{\nu},\kappa)\}^2} \tag{2.19}$$

で与えられる．$c_l(\nu)$ は V 細胞への興奮性固定結合の強度を表し，普通は $|\nu|$ の単調減少関数になるように定められている．この結合範囲 A_l は，S 細胞に至る興奮性可変結合の結合可能領域と同じである．

位置ずれ許容細胞である C 細胞の出力は，次式で与えられる．

$$u_{cl}(\boldsymbol{n},k) = \psi\left[\sum_{\kappa=1}^{K_{sl}} j_l(\kappa,k)\sum_{\nu\in D_l} d_l(\nu)\cdot u_{sl}(\boldsymbol{n}+\boldsymbol{\nu},k)\right] \tag{2.20}$$

ここで，ψ は，C 細胞の入出力間の飽和特性を定める非直線関数で，

$$\psi[x] = \frac{\varphi[x]}{1+\varphi[x]} \tag{2.21}$$

で表される．$d_l(\nu)$ は S 細胞から C 細胞への興奮性固定結合の強度を表し，$c_l(\nu)$ と同じように，ν に関して単調減少関数になるように定められている．D_l はその結合範囲を示している．この式を見ると，前述したように，S 細胞が 1 個でも出力を出せば，この C 細胞も出力を出すことがわかる．

$j_l(\kappa,k)$ は S 細胞の細胞面から C 細胞の細胞面への結合状態を規定する係数で，U_{sl} 層の κ 番目の細胞から U_{cl} 層の k 番目の細胞面に至る結合が存在する場合に値 1 をとり，存在しない場合に 0 をとる．なお，これは主に教師あり信号の場合に意味を持つ項であり，教師なし信号の場合には，$K_{sl} = K_{cl}$ であり，常に

$$j_l(\kappa,k) = \begin{cases} 1 & \kappa = k \text{ のとき} \\ 0 & \kappa \neq k \text{ のとき} \end{cases} \tag{2.22}$$

である．[1]

2.5.2 ネオコグニトンの動作原理

学習については次節で述べるが，ここではネオコグニトロンの学習が終わったとし，どのように入力パターンを認識するのか説明する．

今，ネオコグニトロンが 'A' という文字をすでに学習しているとする． 図 2.10 の上半分に示すように，入力パターン 'A' が入力層 U_0 に与えられると，入力層に近い層の各細胞は，それぞれ小さい受容野で入力パターンを観測し，局所的な特徴を抽出する．その次の層の細胞は，自分の 1 段前の層の細胞が抽出した特徴を，もう少し広い範囲にわたって観測し，いくつかの特徴を統合し，より大局的な特徴を抽出していく．そして，この特徴の抽出と統合の過程で，少しずつ位置ずれを許容していく．

このような操作を繰り返すことにより，最上位層の細胞は入力パターン全体の情報を統合して観測し，特定の入力パターンに対してだけ反応出力を出す． 図 2.10 の場合には，最上位層では，パターン 'A' にだけ対応する細胞 1 個だけが出力を出し，ネオコグニトロンは入力パターンが 'A' であるという答えを出す

さらに詳しく見ると， 図 2.10 の下半分は，U_{s1} 層から U_{c1} 層を経て U_{s2} 層の 1 個の S 細胞 (特徴抽出細胞) に至る細胞間結合状態を拡大して描いた図である．例えば，U_{s1} 層の一番上 ($k=1$) の細胞面には $\bigwedge$ 形の特徴を抽出する細胞が並んでいるが，入力パターン 'A' はその中央上部に $\bigwedge$ の特徴を含んでいるので，この細胞面の中央上部の細胞が反応を示す．これに対し，U_{c1} 層の $k=1$ の細胞面の 1 つの C 細胞は，上記 $k=1$ の S 細胞面の円内に 1 個でも反応している S 細胞があれば，C 細胞も出力を出すように結合されている．つまり，入力パターンの位置ずれにあまり影響されずに，入力層のある範囲内に $\bigwedge$ の特徴があるときに反応出力を出す．

この細胞面にはこにょうな C 細胞が並列的に並んでいるので， 図 2.10 の例ではこの細胞面の中央上部のいくつかの C 細胞が反応出力を出すことになる．U_{c1} 層には，$\bigwedge$ 形の特徴のほかに， 図 2.10 中の $k=2$ や $k=3$ の細胞面に示されるような特徴もある．

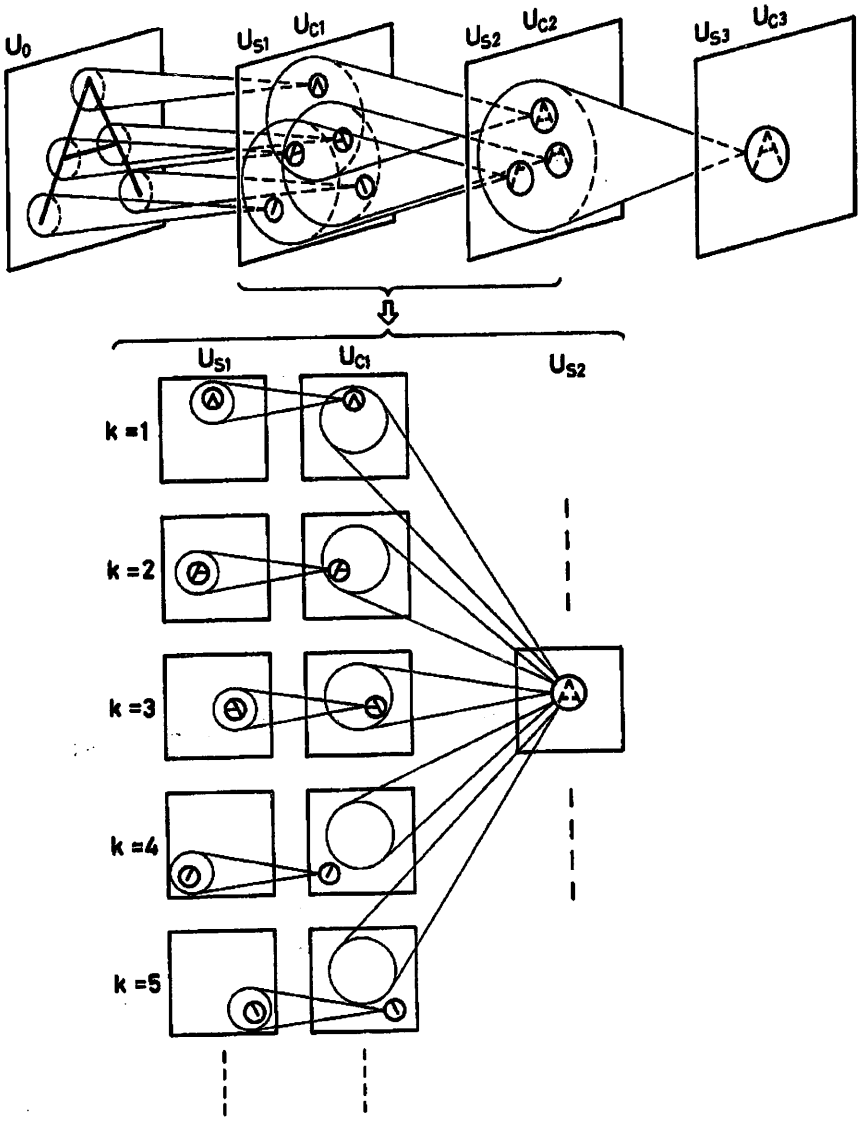


図 2.10: ネオコグニトロンにおけるパターン認識の原理

その次の段のS細胞層であるU_{s2}層の細胞には，たとえば 図2.10の右下方に示すように，U_{c1}に提示された種々の特徴が提示されたときだけ反応するように結合が形成されている．このS細胞は，その受容野に前細胞面 $k=1$，$k=2$，$k=3$，に示されるような特徴がある，という条件を観測しているだけでなく，それ以外の特徴が受容野内に存在しないという条件も同時に確認している．この確認のためには，S細胞に抑制結合をしている抑制性のV細胞が寄与している． [1]

2.6 ネオコグニトロンの学習

2.6.1 重みの更新

ネオコグニトロンの中で可変入力結合をもつのはS細胞だけである．S細胞の入力結合の強化は，最大出力で反応している細胞だけが入力結合の強化を受けるという，最大値検出型仮説に従う．ネオコグニトロンの学習には，最大値検出型仮説のほかに，さらにもう一つ仮説が取り入れられており，最大出力細胞は上記のように自分自身が学習するだけでなく，自分のまわりにあるほかの細胞の入力結合の強化にも影響を与えるという仮説である．つまり，最大出力で反応したS細胞と入力結合と同一の空間分布を持つように成長していく．よって，各細胞面の全細胞がすべて同一の特徴を抽出していくようになっていく．したがって，同一細胞面内の細胞の違いは，特徴を抽出してくる位置だけということになる．

実際にS細胞面内で最大出力を出したS細胞 $u_{sl}(\hat{\boldsymbol{n}},\hat{k})$ が選ばれると，これと同一の細胞面に含まれる細胞への可変結合の強度 $a_l(\nu,\kappa,\hat{k})$ および $b_l(\hat{k})$ は，次式に示す値だけ強化される．

$$\Delta a_l(\nu,\kappa,\hat{k}) = q_l \cdot c_l(\nu) \cdot u_{cl-1}(\hat{\boldsymbol{n}}+\boldsymbol{\nu},\kappa) \tag{2.23}$$

$$\Delta b_l(\hat{k}) = q_l \cdot u_{vl}(\hat{\boldsymbol{n}}) \tag{2.24}$$

ここで，q_lは強化の速度を規定する正の定数であり，この値が大きければ1回の学習で大きく重みを更新することになる．

2.6.2 ネオコグニトロンの学習則

ここでは，ネオコグニトロンで一般的に用いられている，シードセル生成面を用いた学習法 [9] について述べる．

一般的に，第1段目は方位選択性をもつ直線抽出細胞を作りつけてあるので，前節のような重みの更新は，第2段目以降，すなわち，U_{s2}層から上位の層で行われる．学習は低次の層から順に行われ，ある段の結合がすべて決定した状態で，その次の段の学習を行う．

はじめに，学習パターンを入力層に呈示し，学習を行う層の前段までの細胞の反応を求める．次に，入力パターンに含まれる重要な特徴 (局所的特徴) のうち，S細胞でまだ抽出されていない特徴があれば，未学習の細胞面を一つ選び，その細胞面で抽出するように学習を行う．そのために，抽出する特徴の位置にある細胞 (これをシードセルとよぶ) を未学習細胞面内から選び，前段のC細胞の出力に従ってシードセルの入力結合の強化を 式 (2.23)， 式 (2.24) に従って行う．その後，細胞面内のすべての入力結合をシードセルの入

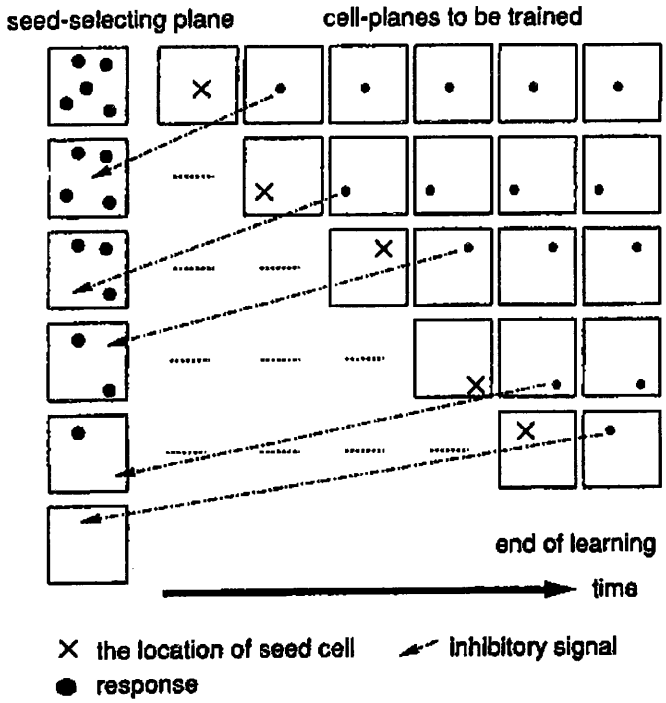


図 2.11: シードセル生成面によるシードセルの選択

力結合と同じ値に変更することにより，細胞面内で同一の空間分布を保つようにする．ここで，このシードセルを選択するための指標となるものとして，シードセル生成面という細胞面を導入する．

シードセル生成面は，入力パターンの重要な特徴のある位置に強い反応を示す細胞面であり，前段のC細胞から固定した入力結合を受け取っている．図2.11に示すように，シードセルはシードセル生成面の中の最大反応を示す細胞の位置に選び，前節に示した学習手法によって未学習の細胞面の学習を行う．次に，学習した細胞面の反応の計算を行い，その反応位置周辺にあるシードセル生成面に対して抑制をかける．その後，再度シードセル生成面内の最大出力細胞位置にシードセルを選んで学習を行う．シードセル生成面内の出力反応が消えるまで，この操作を繰り返し行う．

以上のように自己組織化的にパターンの特徴を抽出するように学習が進み，分類，認識が可能となる．

2.7 2章のまとめ

2章では，ニューラルネットモデルで最も基本的な構成要素である神経細胞とそのモデルについて述べた．また，神経細胞モデルを結合することにより構築されるネオコグニトロン型神経回路モデルについて述べた．ここで述べたように，ネオコグニトロン型モデルは非常に均一な階層構造を持つモデルであり，この階層構造を考慮しつつライブラリを構

成すれば，再利用性や汎用性が高いライブラリが完成することが期待される．

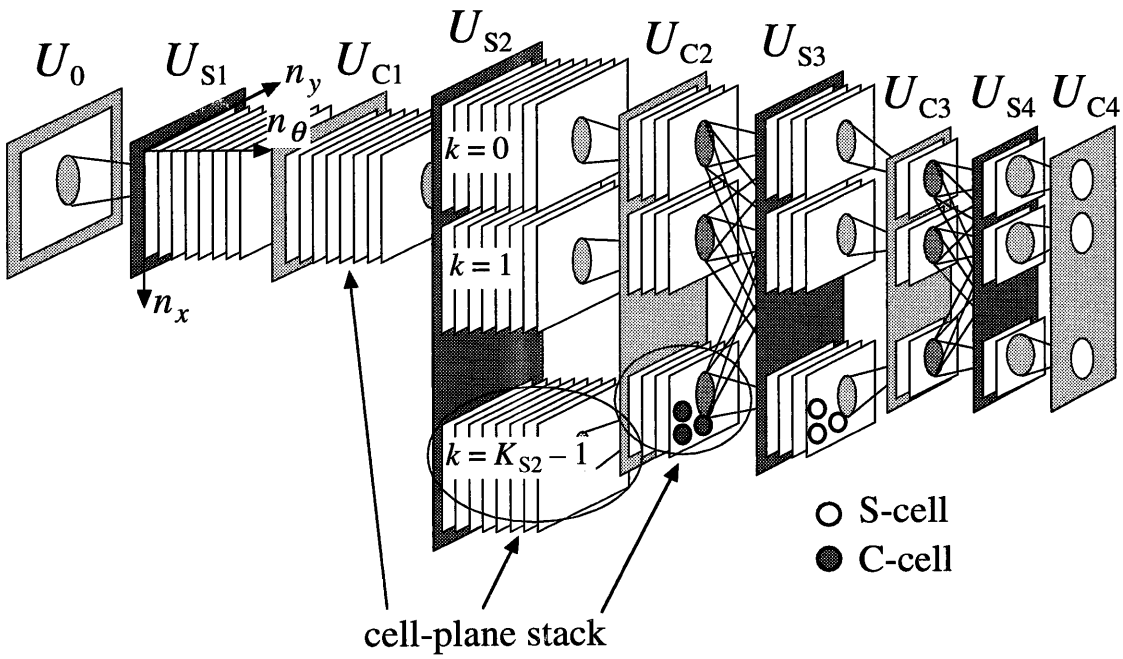


図 2.12: ネオコグニトロンを基礎として発展したモデル例 1
回転したパターンに対応したネオコグニトロン型モデル [3]

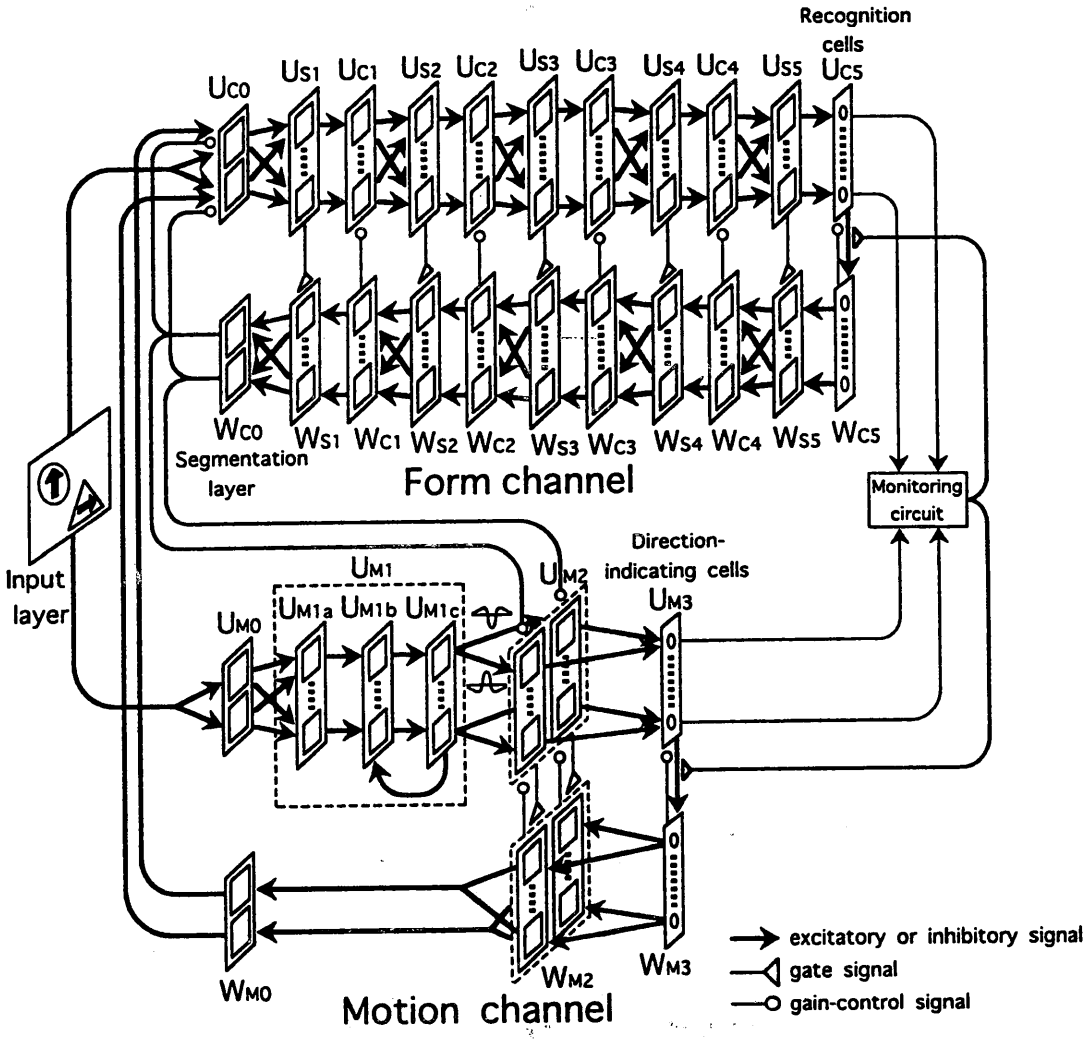


図 2.13: ネオコグニトロンを基礎として発展したモデル例 2
バインディング問題を解決するために提案されたモデル [4]

第3章

ライブラリ設計

この章では，ネオコグニトロン型神経回路モデルの階層的構造を考慮して作成したライブラリについて，その設計方針と，実際の内容について述べる．

3.1 設計方針

本研究で提案するライブラリ (以下，本ライブラリと呼ぶ) において，最も特筆すべき特徴は，本ライブラリに収められているクラス群の構造が，ネオコグニトロン型神経回路モデルの階層的な構造と対応している，という点である．このような設計方針のもとに構築された本ライブラリは，従来になかった汎用性と拡張性を兼ね揃えたものになっており，新しいネオコグニトロン型神経回路モデルや，その他のモデルへ柔軟に対応することが期待される．

第 2 章で述べたように，ネオコグニトロン型神経回路モデルは構造を持った神経回路モデルであり，その構成要素としては，もっとも小さい構成要素である神経細胞からはじまり，複数の細胞により構成される細胞面，さらに，複数の細胞面から構成される細胞層からなっている．

ライブラリを作成するプログラム言語としては，C++言語を選択した．C++はオブジェクト指向技術を取り入れているが，この技術は，特に構造を持ったデータの記述に有効であるとされている．また，C++は，数あるオブジェクト指向言語の中でも多くの分野で使用されており，実行速度にも定評がある．本ライブラリでは，ネオコグニトロン型モデルの構成要素を C++のクラス (class) として実現している．C++を用いて作られたオブジェクト指向プログラムは再利用性や拡張性に富むという意見も多く，本ライブラリには適当なプログラム言語といえる．

3.2 構成要素のクラス

ここでは，実際に作成したライブラリ内のクラス群について順に説明する．

3.2.1 結合

はじめに，細胞をはじめとする各構成要素間の結合を司るクラスについて述べる．
結合を表すクラスは2つあり，

```
template<class Parts> Class OneConnectWeight
template<class Parts> Class ConnectWeight
```

というものである．はじめに Class OneConnectWeight について述べる．

クラス OneConnectWeight はテンプレート引数 class Parts を持つテンプレートクラスであり，データメンバとして次のものを持つ

```
privete:
int nux_, nuy_;
const Parts* connect_;
vector<double>* weight_;
```

nux_，nuy_は自分自身に結合している構成要素との結合範囲を示しており，細胞どうしの結合 (これを以後，細胞間結合とよぶことにする) においては，それぞれの値は1となる．この結合範囲を示す値は，特に細胞面どうしで結合を行う場合 (これを以後，細胞面間結合とよぶことにする) に活用される．

connect_は自分自身に結合している構成要素のポインタを表すデータメンバである．この変数に値を代入することにより，結合が形成される．ここでは代入すべき構成要素へのポインタは Parts 型としてある．Parts 型はテンプレート引数であり，ライブラリ中では後述する Cell 型，Plane 型，Layer 型等として扱われる．

weight_は connect_との結合で用いられる結合重みを代入する変数であり，細胞間結合では普通1つの値の結合重みを持つ．ここでは weight_は C++標準テンプレートライブラリのコンテナの1つである vector として宣言している．vector とは，挿入する要素の数の増加に応じて自動的に格納領域を広げる可変長の配列であり，使用する際には通常の配列と同じように使用することができる [10]．vector により，connect_ 1つに対し結合重み weight_ を複数保持できるように設計したのは，通常，細胞面間結合において細胞面どうしが結合する場合，ある細胞面における1つの細胞は，その細胞が属する細胞面に入力信号を送っている細胞面内の，決められた範囲にある複数の細胞と結合する，という性質があるからである．図3.1にその様子を示す．したがって，細胞面間の結合の場合には，格納する構成要素として細胞面のポインタを connect_ に1つ代入し，weight_ には複数の結合重みを格納する．また，学習則によっては細胞面間の結合範囲が学習過程で変化する場合もあり，そのようなときには weight_ に格納すべき結合重みの値の数が増加，または減少することになる．以上の理由から，weight_ は通常の配列ではなく，vector という可変長配列として宣言してある．

OneConnectWeight についてのデータメンバは以上であり，次にメンバ関数について述べる．

はじめに，コンストラクタについて述べる．

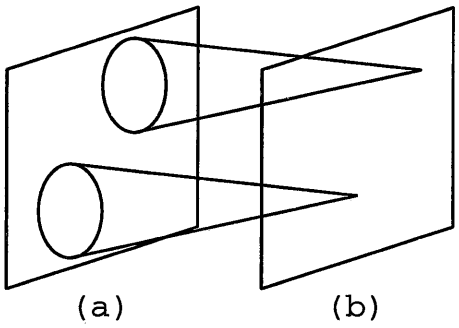


図 3.1: 細胞面間結合の様子

```
OneConnectWeight
 (const Parts& initp, double initw =0.0, int nux =1, int nuy =1)
```

第1引数 initp には自分に結合している構成要素のポインタ，第2引数 initw には重みの初期値，第3引数 nux, および第4引数 nuy には結合範囲の大きさを入力する．これにより，結合範囲は nux×nuy で，それぞれの重みは initw の値で初期化された，initp との結合が形成される．

```
OneConnectWeight
 (const Parts& initp, int nux, int nuy,
                          double(*initwf)(double x, double y))
```

このコンストラクタは，先程のコンストラクタとほぼ同じものであるが，結合重みの初期化に関数を用いることができるようにしたものである．第4引数に double 型の2つの引数を持つ関数へのポインタを代入すれば，たとえばランダムな初期値の設定等に役立つ．

次にコンストラクタ以外のメンバ関数について述べる．これらのメンバ関数が全て virtual としてあるのは，クラスの継承を考慮したためである．

```
virtual int get_numweight(void) const
```

これは現在形成している結合について設定されている結合重みの数を int 型として返すメンバ関数である．

```
virtual const Parts* get_connectptr(void) const
```

これは現在形成している結合のポインタ，すなわち connect_ の値を返すメンバ関数である．

```
virtual double get_weight(int n =0) const
```

結合重み weight_ の値を返す．通常，細胞間結合の場合には結合1つに対して結合重みも1つだけ格納するので，デフォルト引数として weight_の1番目の要素を指定するようにしてある．

```
virtual void add_weight(double inputw, int nux =1, int nuy =1)
virtual void add_weight(int nux, int nuy,
                        double(*initwf)(double x, double y))
```

結合重みを途中から追加するためのメンバ関数．格納する結合重みを nux×nuy と同数増加する．デフォルト引数としては結合重みを 1 つずつ加えるということを意識し，nux=1, nuy=1 とした．1 つ目のメンバ関数は，結合重みの初期化に定数 inputw を用いるものであり，2 つ目のメンバ関数は，結合重みの初期化時に任意の関数を使用することが出来るようにしたものである．

```
virtual void rm_weight(int n)
```

n 番目の結合重みを消去するメンバ関数．weight_は vector であるので，vector のメンバ関数である erase( ) を用いて配列要素を消去する．

```
virtual void set_weight(int n, double inputw)
```

n 番目の結合重みに値 inputw を代入するメンバ関数．

```
virtual void delta_weight(int n, double deltaw)
```

n 番目の結合重みに値 deltaw を加算するメンバ関数．

```
virtual void show_weight(void)
```

格納している結合重みの値を標準出力に列挙するメンバ関数．

```
virtual int get_nux(void) const;
virtual int get_nuy(void) const;
```

それぞれ，結合範囲を表す値，nux_，nuy_の値を返すメンバ関数．

以上で，OneConnectWeight における主なメンバ関数の説明を終わる．

細胞間をはじめとする構成要素間の結合を司るもう 1 つのクラス ConnectWeight は，いわば OneConnectWeight の上位に立つクラスである．このクラスもテンプレートクラスであり，次のようなデータメンバを持つ．

```
private:
int numconnect_;
vector< OneConnectWeight<Parts>* >* connect_;
```

ここで，numconnect_は自分自身に結合している構成要素の数，すなわち，入力結合の数を示している．

connect_ は OneConnectWeight を格納する vector へのポインタ型であり，このクラス ConnectWeight を宣言すると即座に OneConnectWeight 型へのポインタを格納する vector コンテナのオブジェクトが作成されるようになっている．

すなわち，このクラスは，クラス OneConnectWeight が扱っている個々の結合情報を複数データメンバとして持つことにより，複数の結合を扱えるようにしたものである．こ

こで，OneConnectWeight 1つが1つの結合を担当しているので，connect_ の配列要素の数は numconnect_ の値と等しい．

次に，ConnectWeight の主なメンバ関数について記す．このクラスには特別なコンストラクタはなく，デフォルトコンストラクタのみであるのでコンストラクタの説明は省略する．

```
virtual int find_connect(const Parts& parts) const
```

connect_ には複数の OneConnectWeight が格納されるが，このメンバ関数は parts へのポインタを格納している OneConnectWeight が connect_ の何番目に格納されているのか調べ，その値を返すものである．

```
virtual int get_numconnect(void) const
```

numconnect_ の値，すなわち，connect_の配列要素の数を返すメンバ関数．

```
virtual int get_numweight(int n) const
virtual int get_numweight(const Parts& parts) const
```

connect_ に格納されている OneConnectWeight に作成されている結合重みの数を返す．どの OneConnectWeight についての結合重み数かは，上記の1つ目の get_numweight( ) では connect_の配列要素を整数で直接指定して，2つ目の get_numweight( ) では，既に結合を成している Parts 型のオブジェクトを引数に与えてやることで指定する．

```
virtual const Parts* get_connectptr(int n) const
```

n 番目の connect_ に格納されている， OneConnectWeight に作成されている，自分自身に結合している構成要素へのポインタを返すメンバ関数．

```
virtual double get_weight(int m, int n =0) const
virtual double get_weight(const Parts& parts, int n =0) const
```

上記の1つ目の get_weight( ) では connect_の配列要素を整数で直接指定して，2つ目の get_weight( ) では，既に結合を成している Parts 型のオブジェクトを引数に与えることにより指定した OneConnectWeight の n 番目の結合重みの値を返す．デフォルトでは，指定した OneConnectWeight に格納してある結合重みの1番はじめの値を返す．

```
virtual void show_connect(voic  const
virtual void show_weight(void) const
virtual void show_connectweight(void) const
```

結合情報を標準出力に列挙するメンバ関数．show_connect( ) は結合している全ての構成要素へのポインタを列挙する．show_weight( ) は結合している構成要素の全ての結合重みを列挙する．show_connectweight( ) は上記の2つを一度に実行するメンバ関数である．

```
virtual void mk_connect(const Parts& newparts,
                        double initw =0.0, int nux =1, int nuy =1)
virtual void mk_connect(const Parts& newparts, int nux, int nuy,
                        double(*initwf)(double x, double y))
```

結合を形成するためのメンバ関数．mk_connect( ) は構成要素 newparts と結合範囲 nux×nuy で結合し，その結合重みは initw の値で初期化する．

newparts との結合がはじめてであれば，新しく OneConnectWeight を引数で指定した結合条件通りにオブジェクトとして生成し，connect_ へ格納，結合を形成する．はじめてでなければ (既に newparts と結合が生成されていたならば)，newparts との結合を担当している OneConnectWeight で格納している結合重みの数を nux×nuy 分増加させる．上記1つ目の mk_connect( ) は重みの初期化を定数で行うもの，2つ目の mk_connect( ) は重みの初期化を任意の関数で行うものである．

```
virtual void rm_connect(const Parts& rmparts)
```

connect_ に格納されていて，rmparts との結合を担当している OneConnectWeight を削除する．すなわち，rmparts との結合を解除するメンバ関数．

```
virtual void ch_connect(const Parts& oldparts, const Parts& newparts)
```

connect_ に格納されていて，oldparts との結合を担当している OneConnectWeight を，newparts を担当するように変更する．すなわち，oldparts との結合を，newparts への結合へと変更するメンバ関数．

```
virtual void add_weight(const Parts& parts, double input)
```

connect_ に格納されていて，parts との結合を担当している OneConnectWeight の結合重みを増やす．その際に，input の値で結合重みを初期化するメンバ関数．

```
virtual void rm_weight(const Parts& parts, int n)
```

connect_ に格納されていて，parts との結合を担当している OneConnectWeight の n 番目の結合重みを削除するメンバ関数．

```
virtual void set_weight(const Parts& parts, double inputw, int n =0)
```

connect_ に格納されていて，parts との結合を担当している OneConnectWeight の結合重みに inputw の値を代入するメンバ関数．

```
virtual void delta_weight(const Parts& parts, double deltaw, int n =0)
```

connect_ に格納されていて，parts との結合を担当している OneConnectWeight の結合重みに deltaw の値を加算するメンバ関数．

```
virtual int get_nux(const Parts& t_parts) const
virtual int get_nuy(const Parts& t_parts) const
```

connect_ に格納されていて，parts との結合を担当している OneConnectWeight の結合範囲を表す値，nux_，nuy_の値を返すメンバ関数．

以上で，ConnectWeight における主なメンバ関数の説明を終わる．

ここで述べてきたように，結合重みを表すクラスが階層的な設計になっているのは，細胞面 3.2.3 どうしで結合を形成する場合に，計算機資源を無駄に使用しないようにするためである．

細胞面どうしで結合を形成する場合も，もちろん個々の細胞が相手の細胞面内の細胞と結合を形成するわけであるが，ネオコグニトロン型モデルでは，同一細胞面内の細胞が持つ結合重みは，すべて同じ結合状態を持つ，という規則があり，厳密に細胞どうしの結合のみを考えて結合を形成してしまうと，細胞1つ1つが同じ内容の結合重みを持っているにも関わらず，個別にその情報を持つことになり，無駄なメモリを消費してしまうことになる．

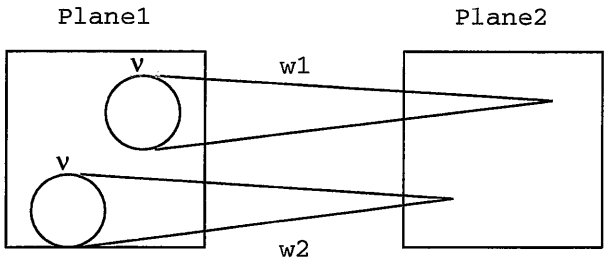


図 3.2: 細胞面間結合の結合重み
$w1$, $w2$ は異なる細胞の結合だが，内容は全く同じ．ν は結合範囲を示す．

図3.2にその例を図示する．細胞面どうしの結合であり，結合範囲 ν と同じ数，結合重みが作成される．個々の細胞でこの結合を形成する場合，図3.2においてPlane2に 10×10 の細胞があるとすれば，$\nu\times10\times10$ の結合重みが必要になってしまう．しかし，実際には ν だけの数の結合重みで細胞面どうしの結合を形成することができる．このように，無駄を省き，計算機資源を有効活用するため，結合を表すクラスを構成要素とは別に作成し，この部分を細胞のみならず他の構成要素にも含ませることにより，細胞のみを考慮した結合では無駄が出てしまうという事態を回避している．

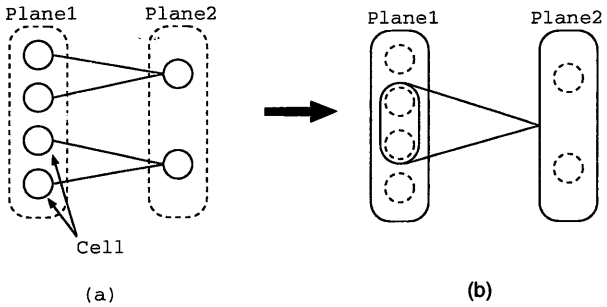


図 3.3: 無駄を省いた結合の概念図
(a) 図は細胞のみの結合で，細胞面間結合を実現した図．
(b) 図は細胞面に結合情報を持たせるようにして，無駄な結合を省いた図．

3.2.2 細胞

神経細胞を司るクラス `Cell` は，次のようなデータメンバを持つ．

```
private:
double celloutput_
ConnectWegihtCell<Cell> connectw_
```

ここで，celloutput_ は細胞の出力値を格納する変数であり，自分自身が他の細胞に出力を送っている場合に，この値が出力値として使用される．

また，connectw_ はテンプレートクラス ConnectWegihtCell<Parts> のオブジェクトであり，これは 3.2.1 で述べた結合情報を司るクラスである．テンプレート引数は Cell としているが，これは細胞クラスは細胞クラスどうしでも結合を成すという前提のもとにこのクラスが設計されているからである．

次に，メンバ関数について説明する．ところで，connectw_ は private として宣言してあるので，これらのメンバにはクラス Cell 以外からのアクセスは許されなくなっている．そのため，connectw_ にアクセスするメンバ関数を Cell クラスのメンバ関数として用意してあるが，その説明は 3.2.1 にある説明と重複するので割愛する．

```
virtual double allconnect_out(void) const
```

これは，自分自身に結合している全ての細胞の出力と重みの積和を出力するメンバ関数である．ただし，ここで求めた出力値は celloutput_ には代入しない．

(～_out( ) 関数は，出力値を celloutput_ に代入しない)

```
virtual double partconnect_out(int n) const
virtual double partconnect_out(const Cell& t_cell) const
```

これらは，自分自身に結合している 1 つの細胞の出力と重みを出力するメンバ関数である．上記の 1 つ目の partconnect_out( ) では自分自身に結合している細胞の指定を整数で直接指定するものであり，2 つ目の partconnect_out( ) では，既に結合を成している細胞のオブジェクトを引数に与えてやることで指定するものである．ここでも，求めた出力値は celloutput_ には代入しない．

```
virtual void set_celloutput(double input)
```

これは，celloutput_ に直接，値 input を代入するメンバ関数である．

```
virtual double output_allconnect(void)
```

これは，自分自身に結合している全ての細胞の出力と重みの積和を出力すると同時に，その出力値を celloutput_ に代入するメンバ関数である．

(output_～( ) 関数は，出力値を celloutput_ に代入する)

```
virtual double output_partconnect(const Cell& cell)
```

これは，自分自身に結合している 1 つの細胞の出力と重みを出力するメンバ関数であり，その細胞は，cell のオブジェクトを引数に与えてやることにより指定する．また，その出力値を celloutput_ に代入する．

```
virtual double output_sigmoid(double th =0.0, double coef =1.0)
```

このメンバ関数は，McCulloch-Pittsのニューロンモデルでよく使用される出力関数，シグモイド関数であり，その式は，

$$f(x) = \frac{1}{1 + \exp\{-a(x - \theta)\}} \tag{3.1}$$

で表される．

th にはしきい値 θ，coef には，式 (3.1) の係数である a の値を引数とする．

以上で，神経細胞を司るクラス Cell の主なメンバ関数の説明を終わる．

3.2.3 細胞面

細胞を複数含む構成要素である細胞面を司るクラス Plane は，次のようなデータメンバを持つ．

```
private:
int plane_x_, plane_y_;
ConnectWeight<Plane> connectw_;

public:
Cell** cell;
```

palne_x_, plane_y_ は，細胞面の大きさの値を格納している変数である．

connectw_ は，3.2.2 節で述べたクラス Cell のものと同様の働きをするデータメンバであるが，ここではテンプレート引数が Plane となっていることからもわかるように，細胞面は細胞面との結合を前提に設計されている．

次のデータメンバ cell は，細胞面には細胞が複数存在し，具体的には細胞が2次元平面状に並べられているので，その実際の構造と全く同じように細胞を2次元配列としてデータメンバとしている．通常，データメンバは private として宣言されることが望まれるが [11]，このデータメンバはライブラリの構造を直感的なものとするため，public メンバとして宣言してあり，クラス外からのアクセスを許可している．

次に，メンバ関数について説明する．

はじめに，コンストラクタを示す．

```
Plane(int px, int py, double initcell =0.0);
```

引数 px, py により，細胞面の大きさを決定する．また，initcell の値により，細胞面内の個々の細胞の初期出力値を決定する．初期状態では全ての細胞がこの出力値を持つことになる．

```
virtual double get_celloutput(int x, int y) const
```

これはデータメンバ cell で，2次元配列 x, y の添字に位置する細胞の出力値を返すメンバ関数である．このメンバ関数において，細胞の位置を指定する際に細胞面の大きさ範囲外の細胞 (plane_x_ < x や，x < 0 の場合) を指定した場合には値0を返す．

```
virtual void set_celloutput(int x, int y, double input)
```

データメンバ cell で，2次元配列 x，y の添字に位置する細胞の出力値を強制的に入力するメンバ関数である．このメンバ関数で細胞面の大きさ範囲外の細胞を指定してしまった場合は値の代入が行われない．

```
virtual int get_planex(void) const
virtual int get_planey(void) const
```

これらのメンバ関数は，細胞面の大きさを表している変数，plane_x_，plane_y_ それぞれの値を返す関数である．

```
virtual Cell* get_cellptr(int x, int y) const
```

データメンバ cell で，2次元配列 x，y の添字に位置する細胞のポインタを返す．

```
virtual void show_plane(void) const
```

cell 個々の細胞の出力値を標準出力に列挙するメンバ関数．

```
virtual void show_cellptr(void) const
```

cell 個々のポインタを標準出力に列挙するメンバ関数．

```
virtual void set_input(double** input)
```

cell 個々の細胞の出力値をまとめて入力するためのメンバ関数．input は，単純に plane_x_，plane_y_ それぞれを配列の要素数とする2次元配列を引数としているもので，これを与えることにより，cell 個々の細胞の出力値が，input の配列のものと等しくなる．

```
virtual void select_seedcell
 (int* x, int* y, double* celloutput) const
```

シードセル生成面を用いた学習法 (2.6.2 節参照) において，シードセル生成面のどの細胞をシードセルとするか，位置の決定を行うメンバ関数．具体的には，最も出力値が大きい細胞をシードセルとして選出する．シードセルの位置を引数 x，y に代入し，出力 (非負) は celoutput に代入する．

```
virtual void print_pgm(char* filename) const
virtual void print_data(char* filename) const
```

細胞面に存在する全ての細胞 cell の出力値をファイルに書き出すメンバ関数．

print_pgm( ) は，“filename” というファイル名で，pgm 形式の画像データとしてファイルに書き出す．このとき，細胞面内の細胞の出力値で最大の値はグレースケールの階調が0に，出力0のものは階調255と変換され出力される．

print_data( ) は，“filename” という名前のファイルに，細胞面内全ての細胞の出力値を列挙して書き出す．

以上で，細胞面クラス Plane のメンバ関数についての説明を終わる．

3.2.4 細胞層

ここでは，細胞面を複数含んでいる細胞層を司るクラス Layer について説明する．ネオコグニトロン型モデルは，通常，細胞層として，S細胞層，C細胞層を構成要素としている．V細胞はS細胞層に含まれる細胞であるが，このライブラリでは，V細胞もV細胞層という1つの細胞層をなし，S細胞層およびC細胞層と結合を形成するものとして扱うこととした．

このクラスは，以下のようなデータメンバを持つ．

```
private:
const int defaultpx_, defaultpy_;
const double defaultinitcell_;
int numplane_;
ConnectWeight<Layer> connectw_;

public:
vector<Plane*> plane;
```

defaultpx_, defaultpy_ および，defaultinitcell_ は，この細胞層に含まれる細胞面の大きさ，および細胞面内の個々の細胞の初期出力値を表すデータメンバであり，この細胞層に含まれる全ての細胞面は，この値に従って生成される．

numpalne_ は細胞層に含まれている細胞面の数を表すデータメンバである．

connectw_ は細胞 (3.2.2 節参照)，細胞面 (3.2.3 節参照) に含まれている同名のデータメンバと同じ働きをするデータメンバである．

plane は細胞面を表すデータメンバである．細胞層には細胞面が複数存在しており，また学習則によってはその数が変化する場合があるので vector 型として宣言し，可変長配列にクラス Plane を格納する設計にした．ここでも，このデータメンバは，クラス Plane における cell のようにライブラリの構造を直感的なものとするため，public メンバとして宣言してあり，クラス外からのアクセスを許可している．

次に，メンバ関数について説明する．細胞層はネオコグニトロン型モデルにおいて最も重要な役割を果たす構成要素であり，数多くのメンバ関数が存在する．

はじめに，コンストラクタについて示す．

```
Layer(int nump, int px, int py, double initcell =0.0)
```

引数 nump により細胞層内における細胞面数を決定する．先に述べたような細胞面数が学習中に変化する場合でも初期細胞面数として値を渡す．px, py はこの細胞層に含む細胞面の大きさを決定する引数であり，initcell には細胞面内の個々の細胞の初期出力値を渡す．この px, py, initcell はそれぞれデータメンバ defaultpx_, defaultpy_, defaultinitcell_ の初期化に用いられ，以後，この細胞層に生成される全ての細胞面はこの値をもとにして構築される．

```
virtual void show_connect_plane(void) const
```

このメンバ関数は，plane に格納されている個々の細胞面における結合重みを標準出力に列挙するメンバ関数である．内部では，ConnectWeight (3.2.1 節参照) のメンバ関数である，connect_weight( ) を細胞面毎に呼出している．

```
virtual int get_numplane(void) const
```

これは，細胞面数を表す numplane_ の値を返すメンバ関数である．

```
virtual Plane* get_planeptr(int nump) const
```

これは，plane の nump 番目に格納されているポインタを返すメンバ関数である．

```
virtual void show_layer(void) const
```

細胞層内の全ての細胞面の個々の細胞の出力値を標準出力に列挙するメンバ関数である．

次に，細胞層内の細胞面が他の細胞層内の細胞面と結合をする場合に使用するメンバ関数を示す．はじめに，ネオコグニトロン型モデルにおいて細胞層どうしの結合方法として代表的なものを図示する．

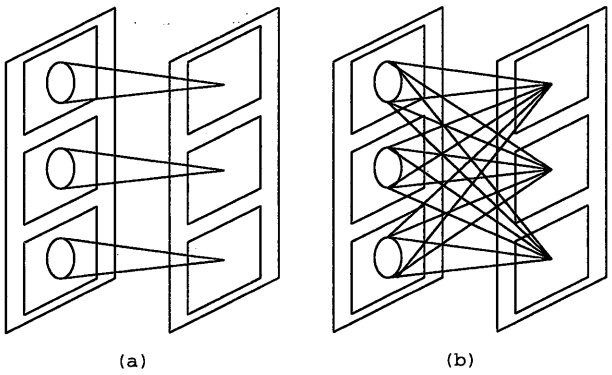


図 3.4: 代表的な細胞層どうしの結合
(a)1 対 1 結合 (b) 多対多結合

図3.4にあるように，細胞層どうしの結合には，大きくわけて 1 対 1 に結合する場合と多対多に結合する場合がある．1 対 1 結合は，S 細胞層から C 細胞層への結合に，多対多結合は，C 細胞層から S 細胞層，V 細胞層から S 細胞層，C 細胞層から V 細胞層への結合で用いられている．

以下に，1 対 1 結合を行うメンバ関数を示す．

```
virtual void mk_1to1connect(const Layer& ps, double initw,
 int nux, int nuy)

virtual void mk_1to1connect_f(const Layer& ps, int nux, int nuy,
 double(*initwf)(double x, double y),
  int(*convergef)(int n) =straight)
```

これらの関数は引数 ps に与えられた細胞層から自分自身の個々の細胞面と 1 対 1 結合を行う．その結合範囲は，nux×nuy である．1 つ目の mk_1to1connect( ) は結合形成時の結合重みの初期値を定数 initw で初期化する．2 つ目の mk_1to1connect( ) は結合形成時の結合重みの初期値を任意の関数 initwf によって行えるようにしたものであり，さらに関数 convergef を引数として与えることにより不規則である結合に対応したものである．この convergef はその引数 n に，今結合させようとしている細胞層の特定の細胞面番号が与えられたときに，返り値として自分自身の特定の細胞層の細胞面番号を返すように設計する．デフォルトでは，straight という引数 n の値をそのまま返す関数を与えており，各々の細胞面は相手の細胞層内の同じ細胞面番号の細胞面と結合するようにしてある．

```
virtual void mk_fullconnect(const Layer& ps, double initw,
         int nux, int nuy)

virtual void mk_fullconnect_f(const Layer& ps,
         int nux, int nuy, double(*initwf)(double x, double y))
```

次に，これらのメンバ関数は引数 ps に与えられた細胞層から自分自身の個々の細胞面と多対多結合を行う．その結合範囲は，nux×nuy である．これらのメンバ関数は，自分自身に含まれる細胞面 1 つずつと，結合相手の細胞層内の全ての細胞面との間に順番に結合を形成していく．1 つ目の mk_fullconnect( ) は結合形成時の結合重みの初期値を定数 initw で初期化する．2 つ目の mk_fullconnect( ) は結合形成時の結合重みの初期値を任意の関数 initwf によって行えるようにしたものである．

上記の mk_1to1connect( ) や mk_fullconnect( ) により，1 対 1 結合や多対多結合に対応したが，これだけを使用して実際にネオコグニトロン型モデルを構築する場合には少々不便である．そこで次に，上記のメンバ関数を利用しつつ，細胞層間の結合をまとめて形成できるメンバ関数について説明する．

```
virtual void mk_Sconnect(const Layer& clayer, double initwc,
 const Layer& vlayer, int numvp, double initwv, int nux, int nuy)

virtual void mk_stdSconnect(const Layer& clayer, double initwc,
 Layer& vlayer, int numvp, double initwv, int nux, int nuy,
  double(*initwf)(double x, double y) =initw1)

virtual void mk_onestdSconnect(int numsp, const Layer& clayer,
 double initwc, const Layer& vlayer, int numvp, double initwv,
  int nux, int nuy)
```

S 細胞層の個々の細胞は 図 2.9(18 頁参照) に示すように，C 細胞，および，V 細胞を介しての C 細胞からの入力を受けとっているが，これらのメンバ関数を S 細胞層へ実行すれば，以上の C 細胞から S 細胞，C 細胞から V 細胞，V 細胞から S 細胞への結合を容易に形成することができる．

mk_Sconnect( ) は，C 細胞層から S 細胞層への結合，および，V 細胞層から S 細胞層への結合を形成するメンバ関数である．

mk_stdSconnect( ) は標準的なC細胞層からS細胞層への結合を一手に担うメンバ関数で，mk_Sconnect( ) および，後述の mk_stdVconnect( ) を併用して，C細胞からS細胞，C細胞からV細胞，V細胞からS細胞への結合全てを形成する．

mk_onestdSconnect( ) は mk_stdSconnect( ) が全てのS細胞層内の細胞面を対象に結合を形成するのに対し，S細胞層内の特定の細胞面について，前層のC細胞層，および，V細胞層との結合を形成するメンバ関数である．

これらのメンバ関数で用いられる引数については，clayer，vlayer はそれぞれ結合を形成したいC細胞層，V細胞層を引数として与える．また，V細胞層に関しては引数 numvp へ結合に使用する細胞面番号を入力しなくてはならない．この結合が形成される際の，C細胞からS細胞への結合，および，V細胞からS細胞への結合，といった可変結合の部分は，その重みの初期値として initwc，initwv に値を与える．また，結合範囲は nux×nuy で指定する．通常，C細胞層からV細胞層への結合範囲，および，C細胞層からS細胞層への結合範囲は同じものであるので，これらの結合範囲にはともにこの値を用いる．mk_stdSconnect( ) にある関数へのポインタとしての引数 initwf には，C細胞層からV細胞層の固定結合の重みを決定する関数を与える．デフォルトでは，initw1 という関数が与えられるが，この関数は呼出されても値1を返すことしかしない関数である．

以上，3つの関数について，その担当する結合の様子を 図3.5に図示する．

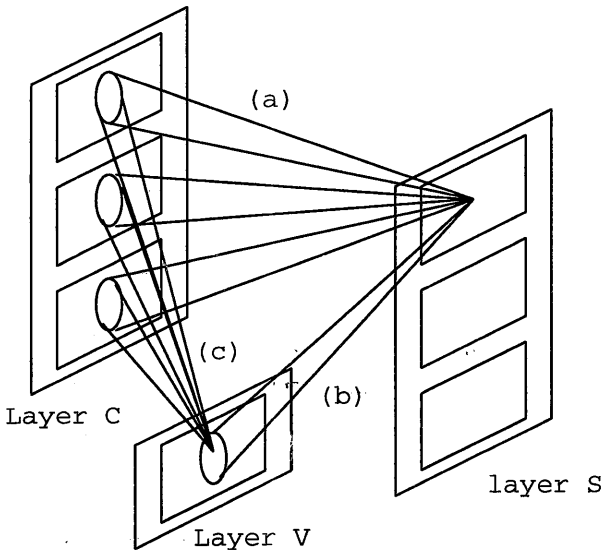


図 3.5: S細胞からC細胞への結合に使用されるメンバ関数

mk_Sconnect( ) は (a),(b) の結合を，mk_stdSconnect( ) は (a),(b),(c) の結合をS細胞層内全ての細胞面に，mk_onestdSconnect( ) は (a),(b),(c) 結合を図のように1つの細胞面に対して形成する．

次に，C細胞層の結合に関するメンバ関数について説明する．

```
virtual void mk_stdCconnect(const Layer& slayer, int nux, int nuy,
 double(*initwf)(double x, double y) =initw1,
```

```
int(*convergef)(int n) =straight)
```

このメンバ関数はS細胞層からC細胞層への結合を全ての細胞面について形成するものであり，実際には 38 頁で述べた mk_1to1connect_f( ) と同様のふるまいをする．引数としては slayer には前層のC細胞層を与える．他の引数については前述した mk_Sconnect( ) をはじめとした，S細胞層の結合に使用するメンバ関数と同様である．また，引数 convergef についても 38 頁で述べたものと同様である．

次に，V細胞層の結合に関するメンバ関数について説明する．

```
virtual void mk_stdVconnect(const Layer& clayer, int nux, int nuy,
 double(*initwf)(double x, double y) =initw1)

virtual void mk_onestdVconnect(int numvp, const Layer& clayer,
 int nux, int nuy, double(*initwf)(double x, double y) =initw1)
```

mk_stdVconnect( ) はC細胞層からV細胞層への結合の全てを形成するメンバ関数である．このメンバ関数はV細胞層内に複数のV細胞面が存在した場合，それら全てが前層のC細胞層内の細胞面と結合する．ネオコグニトロン型モデルの場合，通常，V細胞面は1つしか使用しないので，V細胞層に細胞面が複数作成してあったとしても，その内1つのみを使用することになると思われる．そのために用意したのが，mk_onestdVconnect( ) であり，このメンバ関数はV細胞層内で引数 numvp 番目の細胞面のみを前層のC細胞層と結合させる．

以上で，各細胞層間の結合を行うメンバ関数の説明を終わる．

次に，細胞面の数を制御するメンバ関数について述べる．

```
virtual int add_plane(void)
virtual void del_plane(int num_p)
```

add_plane( ) は細胞層内の細胞面を増加させ，del_plane( ) は細胞層内の細胞面で引数 num_p 番目のものを削除する．前述したように，add_plane( ) により細胞面を増やす際にはデータメンバに蓄えられている情報をもとに細胞面を作成するので，このメンバ関数では細胞面についてのデータを引数として与えなくて済むようにしてある．また，細胞面増加時には返り値として増加した細胞面の細胞面番号を返す．

次に，細胞層の個々の細胞面について出力計算を行うメンバ関数について述べる．

```
virtual void output_Slayer(const Layer& clayer,
 const Layer& vlayer, int numvp, double r, double sigma =1.0)

virtual void output_stdSlayer(const Layer& clayer, Layer& vlayer,
 int numvp, double r, double sigma =1.0)
```

これらは，標準的なネオコグニトロン型モデルのS細胞層の出力計算を行うメンバ関数である．

output_Slayer( ) は 式 (2.18)(18 頁参照) に従って，C 細胞層から S 細胞層への結合，および，V 細胞層から S 細胞層への結合を用いて S 細胞層の個々の細胞の出力値を計算するものである．

また，output_stdSlayer( ) は内部で上記の計算をするのに先立ち，後述する output_stdVlayer( ) を実行し，V 細胞層の個々の細胞の出力を求めるため，S 細胞層の出力を求める際に V 細胞層の出力計算を意識することなく行うことができる．

引数 r には，抑制入力を制御する定数である 式 (2.18) の r_l に相当する値を与える．

ところで， 式 (2.18) は，古典的なネオコグニトロンでは，

$$u_{sl}(\boldsymbol{n},k) = r_l \cdot \varphi \left[\frac{\sigma_l + \sum_{\kappa=1}^{K_{cl}-1} \sum_{\nu \in A_l} a_l(\boldsymbol{\nu},\kappa,k) \cdot u_{cl-1}(\boldsymbol{n}+\boldsymbol{\nu},k)}{\sigma_l + \dfrac{r_l}{1+r_l} \cdot b_l(k) \cdot u_{vl}(\boldsymbol{n})} - 1 \right] \tag{3.2}$$

としていた．この 式 (3.2) では，σ_l というパタメータが存在している．しかし，現在では，この値は特にパラメータとせず，定数 1 とするのが常識である．このような古典的なモデルにも対応すべく，これらのメンバ関数では引数 sigma に σ_l の値を与えることができるように作成した．しかし，以上のような背景からデフォルト引数として値 1.0 を与えている．

次に，

```
virtual void output_stdClayer(const Layer& slayer)
```

このメンバ関数は C 細胞層の個々の細胞の出力を行うものであり，その計算は 式 (2.20)(18 頁参照) に従って行われる．

また，

```
virtual void output_stdVlayer(int nump, const Layer& clayer)
```

このメンバ関数は V 細胞層の個々の細胞の出力を行うものであり，その計算は 式 (2.19)(18 頁参照) に従って行われる．引数 nump には，V 細胞層内の細胞面で使用する細胞面番号を与える．

ここで述べた種々の細胞層の出力計算用のメンバ関数を使用すると，その結果は細胞層内の細胞面にある細胞の出力値として，その値が保存される．具体的には，クラス Cell のデータメンバである celloutput_に値が代入される．また，引数としてクラス Layer 型の値を必要とするものが多いが，このような細胞層との間には，前述した mk_stdSconnect( ) 等を用いて然るべき結合状態が形成されていることが前提である．

次に学習時に用いるメンバ関数について述べる．

```
virtual Plane* mk_ssp(const Layer& clayer, int nux, int nuy) const
```

このメンバ関数は，シードセル生成面 (Seed-Selecting Plane) を作成するためのものである．シードセル生成面は，S 細胞層において学習させるべき細胞を選出するための細胞面であり，前層の C 細胞層から固定結合を受けと取っているため，引数 clayer に C 細胞層，nux，nuy にその固定結合の結合範囲を示す値を与えなければならない．

```
virtual void response_stdSlayer
 (int x, int y, const Layer& clayer, Layer& vlayer, int numvp,
  double r, double sigma =1.0)
```

このメンバ関数は，S細胞層内の全ての細胞面について，ある位置 x，y だけの出力計算を行うものである．実際に使用する場合は，ある位置 x，y にシードセルとして選ばれた細胞の位置を渡す．

```
virtual int find_max_firing_plane(int x, int y) const
```

このメンバ関数は，細胞層内の細胞面の特定の位置，x，y の出力を全ての細胞面で比較し，その中で最も大きい出力を出している細胞面番号を返すものである．もし，どの細胞面でも，位置 x，y が出力を出していない (出力が 0) ならば，−1 を返す．シードセルを選出した後，どの細胞面に学習を施すのか決定する場合や，最終層の出力状況を確認する場合に使用する．

次に，学習時に重みを実際に変更するメンバ関数について述べる．

```
virtual void learning_excite
 (int nump, const Layer& prvclayer, int cx_index, int cy_index,
  double q, double(*weightc)(double x, double y));
```

learning_excite( ) は，式 (2.23) (21 頁参照) の式に従って，興奮性結合の重みの更新を行う．引数 nump には学習を施す細胞面番号，prvclayer には，前層の C 細胞層を与える．cx_index，cy_index は，学習を施す細胞の位置，すなわちシードセルの位置，を与える．q は学習速度を決定する定数である．また，weightc には，C 細胞層から S 細胞層への固定結合を定義している関数を与える．

```
virtual void learning_inhibit
 (int nump, const Layer& vlayer, int numvp,
  int vx_index, int vy_index, double q);
```

learning_inhibit( ) は，式 (2.24) (21 頁参照) の式に従って，抑制性結合の重みの更新を行う．引数 nump には学習を施す細胞面番号を与える．vlayer には，学習に使用する V 細胞層を与え，さらに numvp，vx_index，vy_index により，V 細胞層内で学習に使用する細胞面番号と細胞の位置を与える．q は学習速度を決定する定数であり，learning_excite( ) で使用するものと同じ値である．

以上で，クラス Layer の学習に関するメンバ関数の説明を終わる．

次に，細胞層内の細胞面の状態をファイルとして出力するメンバ関数について述べる．

```
virtual void printall_pgm(char* name, int numx,
 int index1 =0, int index2 =0) const;

virtual void printall_data(char* name,
 int index1 =0, int index2 =0) const;
```

```
virtual void printall_pgm_data(char* name, int numx,
 int index1 =0, int index2 =0) const;
```

printall_pgm( ) は，“name_index1_index2.pgm” というファイル名に，全ての細胞面の状態を，pgm 形式の画像データとしてファイルに書き出すメンバ関数である．numx には，この画像データの横方向にいくつの細胞面を並べて表示するのか，その数を与える．

printall_data( ) は，“name_index1_index2.txt” というファイル名に，全ての細胞面の状態を列挙するメンバ関数である．

printall_pgm_data( ) は以上 2 つのメンバ関数を 1 度に実行するメンバ関数である．

以上で，細胞層クラス Layer についての説明を終わる．

3.3 学習則

2.6.2 節で述べたように，本ライブラリにはネオコグニトロン型モデルにおいて最も一般的であるシードセル生成面を用いた学習則を含んでいる．これは，次のような関数として作成してある．

```
void learningssp
 (const Layer& prvclayer,
  Layer& slayer, double initwa,
   Layer& vlayer, double initwb, int prvctosnux, int prvctosnuy,
    Layer& clayer, int stocnux, int stocnuy,
     double q, double r,
      double(*initwc)(double x, double y) =initw1,
       double(*initwd)(double x, double y) =initw1)
```

引数 prvclayer には前層の C 細胞層を与える．slayer，vlayer，clayer，にはそれぞれ，学習を行う S 細胞層，V 細胞層，C 細胞層を与える．

initwa，initwb は S 細胞層に結合を成す興奮性 (C 細胞層から S 細胞層)，および，抑制性 (V 細胞層から S 細胞層) の可変結合の初期値を定数で与える．initwc は前層の C 細胞層から V 細胞層に，initwd は S 細胞層から C 細胞層に結合を成す，それぞれ興奮性の固定結合の結合重みを任意の関数である．デフォルト引数で与えている関数 initw1( ) はどのような引数に対しても 1 を返す関数である．

prvctosnux および prvctosnuy には，前層の C 細胞層から S 細胞層への結合範囲，sctocnux および stocnuy には，S 細胞層から C 細胞層への結合範囲を指定する．なお，前層の C 細胞層から V 細胞層への結合の範囲は，前層の C 細胞層から S 細胞層への結合範囲と等しいので個別に入力する必要はない．

また，引数 q には学習定数を，r には抑制定数を与える．

このような種々の引数から，slayer，clayer へ入力信号を送っている結合重みや細胞面数が，学習則に従って更新される．

3.4 3章まとめ

この章では，ネオコグニトロン型モデルを対象としたライブラリについて提案した．このライブラリの大きな特徴はネオコグニトロン型モデルの階層構造と，ライブラリの構造が一致しているという点である．そのため，直感的なライブラリ操作が可能となっている．また，各構成要素が階層構造を成しているため，学習時におけるモデルの動的な構造変化にも対応することができた．さらに，構成要素間の結合を形成する際には，無駄な計算機資源を使用しないように工夫を施した．

ここに，ライブラリ中のクラスの全体図を示す．

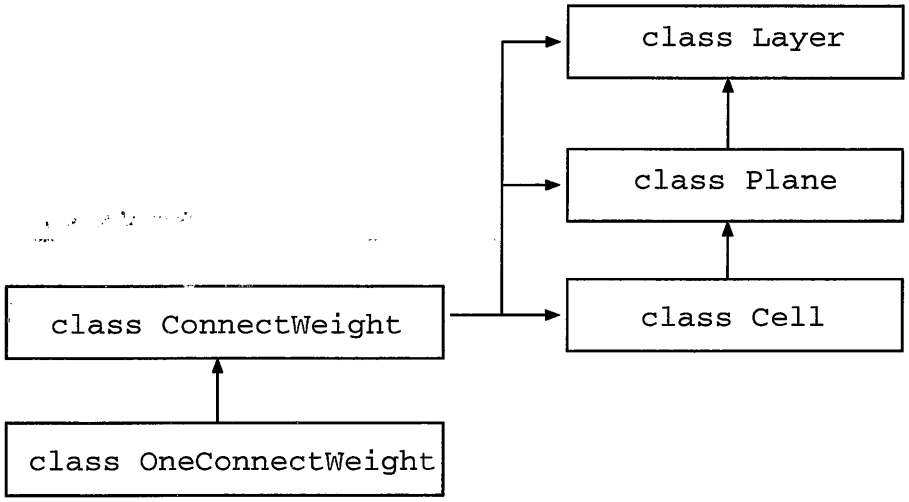


図 3.6: ライブラリ中のクラスの相関図
図中 ← は，クラスをデータメンバとして含むことを示している．

ライブラリのソースコード量は，

細胞に関して 255 行

細胞面に関して 401 行

細胞層に関して 921 行

結合に関して 510 行

学習に関して 117 行

その他 60 行

合計 2206 行

以上のとおりである．

第4章

ライブラリの評価

この章では，実際にネオコグニトロンを本ライブラリを用いて構築し，その挙動やプログラミング量から，本ライブラリの有効性を説明する．

4.1 モデル設計

本研究では，最も基本的なネオコグニトロン [2] をもとにしてモデルを作成した．
入力層 U_0 は 19×19 ，
第 1 S 層 U_{s1} は 19×19×12 ，
第 1 C 層 U_{c1} は 21×21×12 ，
第 2 S 層 U_{s2} は 21×21×K_{s2} ，
第 2 C 層 U_{c2} は 13×13×K_{c2} ，
第 3 S 層 U_{s3} は 13×13×K_{s3} ，
第 3 C 層 U_{c3} は 7×7×K_{c3} ，
第 4 S 層 U_{s4} は 3×3×K_{s4} ，
最終層である第 4 C 層 U_{c4} は 1×1×K_{c4} とした．

ここで，$X \times Y \times K$ は，細胞面の大きさ縦 × 細胞面の大きさ横 × 細胞面数，を表しており，K_{sl} および $K_{cl}(l = 2, 3, 4)$ は学習により変化する細胞面数を表している．このモデルの様子を 図 4.1 に示す．

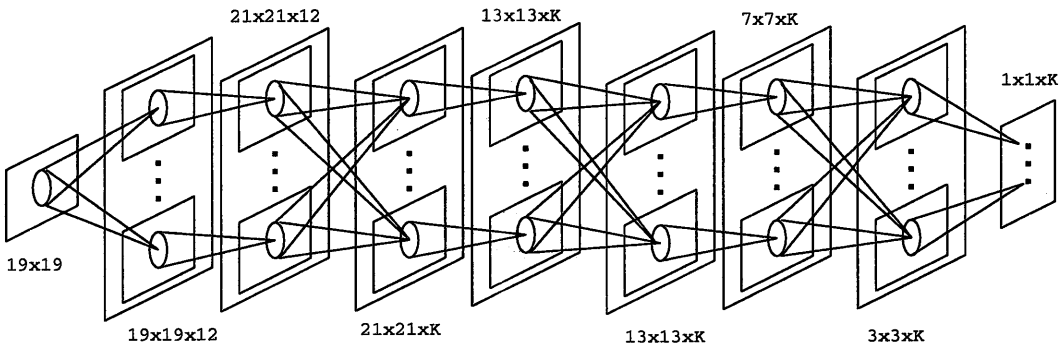


図 4.1: 作成したネオコグニトロンの図

次に，本ライブラリを使用してモデルを作成する例を順を追って説明する．

はじめに，ネオコグニトロンの構造を決定する．ネオコグニトロンは複数の細胞層を持ったモデルであるので，細胞面のクラスを複数宣言することになる．

```
//入力層
Layer C0(1, 19,19);
//第1段
Layer V1(1, 19,19);
Layer S1(12, 19,19);
Layer C1(12, 21,21);

//第2段
Layer V2(0, 21,21);
Layer S2(0, 21,21);
Layer C2(0, 13,13);

//第3段
Layer V3(0, 13,13);
Layer S3(0, 13,13);
Layer C3(0, 7,7);

//第4段
Layer V4(0, 3,3);
Layer S4(0, 3,3);
Layer C4(0, 1,1);
```

この例のように，細胞層の構造決定をこれだけで終わらせることができる．第2段から第4段まで (U_{s2} から U_{c4} まで) の細胞層内の細胞面数 (第1引数) は 0 としているが，これは，はじめは細胞層内に細胞面が存在せず，学習の進行に伴って増加するためである．

次に，細胞層間の結合を決定する．結合に関しても，そのほとんどが学習過程に行われる．しかし，第1段まで (U_{c1} まで) の結合はあらかじめ次のように指定する．

```
S1.mk_stdSconnect(C0, 0.1, V1, 0, INITB, 3,3);
set_initw1(C0, S1);
C1.mk_stdCconnect(S1, 3,3);
```

これで，第1段の結合に関しては，その全てが形成される．S1.mk_stdSconnect( ) により，入力層 U_0 と 第1S層 U_{s1} との結合が．V細胞の結合も含め完了する．また，C1.mk_stdCconnect( ) により，第1S層 U_{s1} と 第2C層 U_{c1} との結合が完了する．第1段においては結合重みを手動で与えるが，ここでは set_initw1( ) がその役割りを果しており，この関数により U_0 と U_{s1} の間の興奮性の結合重みが入力される．また，このときに抑制性の結合も手動で与えるが，その値は INITB に入力する．この結合の例では，S細胞層，C細胞層は共に 3×3 の結合範囲で前層と結合している．

以上で，学習までの準備は終了である．このあと，入力層にパターンを呈示し，学習則に沿ってネオコグニトロンに学習を施し，その後，認識実験へと移る．

```
//1段目-----------------------------------------------------
S1.output_stdSlayer(C0, V1, 0, R1);
C1.output_stdClayer(S1);
```

これは，1段目 (U_{s1}, U_{c1}) の計算の実行例である．1段目は先に述べたようにあらかじめ結合重みが決定されているので，学習をする必要はなく，呈示されたパターンについて出力計算を行うのみである．

```
//2段目-----------------------------------------------------
if( stage == 0 ){
  learningssp(C1, S2, 0.1, V2, 0.1, 5,5, C2, 7,7, 100000, R2);
}
else{
  S2.output_stdSlayer(C1, V2, 0, R2);
  C2.output_stdClayer(S2);
}
```

次に，2段目 (U_{s2}, U_{c2}) の例を示す．実際にはこの段から学習が行われる．learningssp( ) は本ライブラリに含まれるシードセル生成面を用いた学習法を実行する関数である．ここでは，前層のC細胞層，学習を行うS細胞層，V細胞層，および，C細胞層，各層の結合範囲，可変結合重みの初期値，学習定数など，多くの引数を必要とするが，この関数のみで1段ぶんの学習を行うことができる．ところで，ここに if 文が記述してあるのは，第3段の学習の際には前段である第2段の出力結果を使用するので，このように学習の進行状態に沿って分岐するプログラムとなっている．

以上を最終層まで繰り返し行い，全ての細胞層間の結合重みが決定されたならば学習は終了する．

学習が終了すると，次は認識過程に移る．ここでは，入力層に評価用のパターンを呈示し，全ての層の出力を計算することが要求される．実際のプログラムでは次のように記述する．

```
C0.plane[0]->set_input(d_data);//入力パターンの呈示

//第1段目
S1.output_stdSlayer(C0, V1, 0, R1);
C1.output_stdClayer(S1);
```

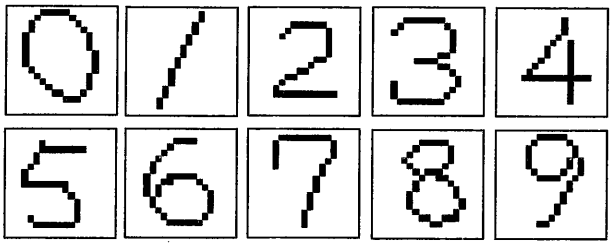


図 4.2: 学習に用いたパターン

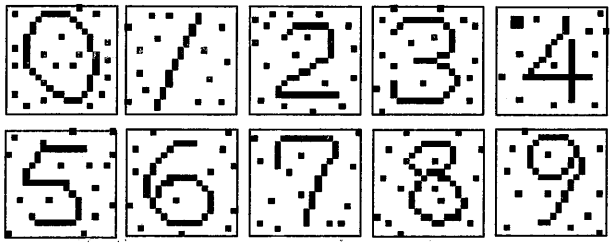


図 4.3: 認識実験に用いた評価用パターン

```
//第2段目
S2.output_stdSlayer(C1, V2, 0, R2);
C2.output_stdClayer(S2);

//第3段目
S3.output_stdSlayer(C2, V3, 0, R3);
C3.output_stdClayer(S3);

//第4段目
S4.output_stdSlayer(C3, V4, 0, R4);
C4.output_stdClayer(S4);
```

このように，各層における出力を繰り返し記述するだけで終了する．

以上でネオコグニトロンの動作についての記述を終わるが，本ライブラリを用いてモデルを作成した場合，このような少い量で，モデルの構築，学習，および認識実験ができる．これは，本ライブラリが，モデル構築時のプログラミング量の削減に大きく貢献しているといえる．

4.2 認識実験

次に，本ライブラリの挙動を検証するため，実際に学習用パターンをモデルに呈示，学習した後に認識実験を行った．今回の実験では，図4.2に示すような，0から9の数字を用いた．大きさは19 × 19ピクセルのpgm形式の画像である．また，認識実験の際に呈示する評価用のパターンは，図4.2の他にも図4.3に示すような，ノイズが混っているパ

ターンも用いた．この実験では，すべての評価パターンに対し最終層の正しい細胞が発火し，パターンの分類に成功した．

4.3 拡張されたモデルへの対応

ネオコグニトロン型モデルをもとに発展したモデルは，その構成要素が一部拡張されたものになっている場合がある．代表的な例としては 図2.12に示したモデルがあげられる．このモデルには，細胞面群という，細胞面を3次元に拡張した構成要素が用いられている．

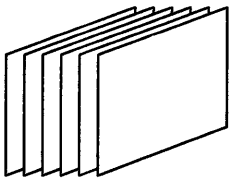


図 4.4: 拡張された構成要素：細胞面群

図4.4にその図を示す．この構成要素は，細胞面を複数持っているという，構造としては細胞層と同じものである．したがって，この構成要素を表すクラスは細胞層のクラスを継承することにより作成することができる．その記述例を以下に示す．

```
#include "Layer.h"

class PlaneStack : public Layer
{
public:
  PlaneStack(int nump, int px, int py, double initcell =0.0);

  virtual void show_planestack(void) const;
};
```

さらに，このモデルには細胞面群を伴う細胞層が用いられている 図4.5.

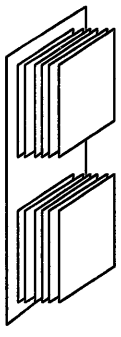


図 4.5: 拡張された構成要素：細胞面群を伴った細胞層

この構成要素は細胞層と細胞面群を組合せることにより作成することができる．以下にその記述例を示す．

```
class LayerPS
{
private:
  int numplanestack_;
  ConnectWeight<LayerPS> connectw_;

public:
  vector<PlaneStack*> planestack_; //細胞面群

//以下メンバ関数
```

以上のように，比較的楽に新しい構成要素を作成できる．

4.4 ネオコグニトロン型モデル以外のモデルへの応用

次に，このライブラリをネオコグニトロン以外のモデルに使用する例を示す．

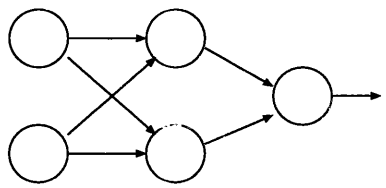


図 4.6: 簡単な 3 層パーセプトロン

図 4.6 のような 3 層パーセプトロンの構造，および結合をプログラミングする場合，以下のように記述する．

```
Cell input_unit[2];
Cell hidden_uit[2];
Cell output_unit;

//結合形成
for(int i=0; i<2; i++){
  for(int j=0; j<2; j++){
    //入力層→隠れ層
    hidden_unit[i].
     mk_connect(input_unit[j], rand()/(double)RAND_MAX);
  }
  //隠れ層→出力層
  output_unit.
     mk_connect(hidden_unit[i], rand()/(double)RAND_MAX);
}
```

これだけの量で，構造，結合については記述が終了し，モデルの構築が完成する．

4.5 4章のまとめ

4章では，作成したライブラリを用いて実際にネオコグニトロンを作成し，認識実験を行った．その結果，ノイズが混っているパターンを正しく認識することができ，ライブラリが正確な動作をしていることを示した．

また，拡張された構成要素を作成する場合も，細胞面や細胞層と同じ内部構造で作成することができる例を示した．

さらに，ネオコグニトロン以外のモデルにも使用できる例として，パーセプトロンの記述例を示した．

第5章

結論

5.1 本研究の成果

本研究では．視覚系神経回路モデルの中でも，パターンの変形，位置ずれに頑健なネオコグニトロン，およびネオコグニトロンを基礎として発展したモデルに注目し，これらのモデルを容易に構築，プログラミングするための開発支援システムの作成を目的とした．

第2章では，神経生理学等から得られた結果をもとにして作成され，工学的にも有効性が期待される視覚系神経回路モデル，ネオコグニトロンについて，その最も基本的な構成要素である神経細胞のモデル化からはじまり，細胞面，細胞層，そしてネオコグニトロンについて，その構造および動作原理について述べた．また，学習則については，最も一般的な学習則である，シードセル生成面を用いた学習法について述べた．

第3章では，ネオコグニトロン型モデルを容易に構築，プログラミングすることを目的としたC++クラス群からなるライブラリを提案した．ここで作成したライブラリは，ネオコグニトロン型モデルの階層構造と一致した構造を持っているのが特徴である．これにより，ライブラリの直感的な操作が可能であり，また，モデルの動的な構造変化にも対応している．

第4章では，第3章で作成したライブラリを使用してネオコグニトロンを構築する例を示し，構造，学習，認識実験における全てのプログラミング量が少い記述で済むことを示した．また，この例で示したネオコグニトロンを実際に動作させ，パターンの学習，認識実験を行い，通常のパターン以外にノイズが混ったパターンをも正しく認識することを示した．これにより，本ライブラリが正しく動作していることを証明した．

5.2 今後の課題

本研究で提案したライブラリの特徴は，ネオコグニトロン型モデルと同じ階層構造を実現している点であると述べた．本ライブラリは，構成要素のみならず，それらを結ぶ「結

合」に関する部位も階層構造を成している．そのために，モデルの動的な構造変化への対応や，計算機資源の有効利用を可能としたが，ある構成要素において，その深い部分へのアクセスが非常に面倒なものとなっている．たとえば，細胞層クラスにおいて，ある細胞層が所有している個々の細胞の結合を形成させる場合などは，「細胞層→細胞面→細胞・結合のためのメンバ関数」というように，長い記述をしなければならない．これは確かに直感的にわかり易いといえるが，本研究の目的であるプログラミング量の削減には相反する．

また，今回第4章では，最も古典的かつ標準的なネオコグニトロンによりライブラリの評価，検証を行ったが，より多くのネオコグニトロン型モデル (たとえば，回転対応型ネオコグニトロン [3] 等) についても本ライブラリが有効であることを示すことが必要であると考える．

付録 A

ライブラリソースリスト

```
// -*-c++-*-
// Neuron OneConnectWeight Class

#ifndef ___ONECONNECTWEIGHT_H___
#define ___ONECONNECTWEIGHT_H___

#include "n_vector.h"

template <class Parts>
class OneConnectWeight
{
private:
  bool external_;
  int nux_, nuy_;
  const Parts* connect_; // do not create array.
  n_vector<double>* weight_;

public:
  OneConnectWeight(const Parts& initp, double initw =0.0, int nux =1, int nuy=1);
  OneConnectWeight(const Parts& initp, int nux, int nuy, double(*initwf)(double x, do
uble y));
  OneConnectWeight(const Parts& initp, const OneConnectWeight<Parts>& initwptr);
  OneConnectWeight(const OneConnectWeight<Parts>& x, const Parts& initp);
  virtual ~OneConnectWeight(void);
  OneConnectWeight(const OneConnectWeight& x);
  OneConnectWeight& operator=(const OneConnectWeight& x);

  // methods for weight
  virtual int get_numweight(void) const;
  virtual const Parts* get_connectptr(void) const;
  virtual n_vector<double>* get_weightptr(void) const; // for weight sharing
  virtual double get_weight(int n =0) const;
  virtual void add_weight(double inputw, int nux =1, int nuy =1);
  virtual void add_weight(int nux, int nuy, double(*initwf)(double x, double y));
  virtual void rm_weight(int n);
  virtual void set_weight(int n, double inputw);
  virtual void delta_weight(int n, double deltaw);
  virtual void show_weight(void);

  // dummy methods for OneConnectWeightPlane :-p
  virtual int get_nux(void) const;
  virtual int get_nuy(void) const;
};

// methods for OneConnectWeight
template <class Parts>
inline OneConnectWeight<Parts>::OneConnectWeight(const Parts& initp, double initw, in
t nux, int nuy)
  :external_(0), nux_(nux), nuy_(nuy), connect_(&initp), weight_(new n_vector<double>
)
{
  for(int i=0; i<nux; i++){
    for(int j=0; j<nuy; j++){
      (*weight_).add_element(initw);
    }
  }
}

template <class Parts>
inline OneConnectWeight<Parts>::OneConnectWeight(const Parts& initp, int nux, int nuy
, double(*initwf)(double x, double y))
  :external_(0), nux_(nux), nuy_(nuy), connect_(&initp), weight_(new n_vector<double>
)
{
  int cx = nux/2; //index of center nux
  int cy = nuy/2; //index of center nuy
```



```
  for(int i=0; i<nux; i++){
    for(int j=0; j<nuy; j++){
      (*weight_).add_element(initwf((double)(i-cx), (double)(j-cy)));
    }
  }
}

template <class Parts>
inline OneConnectWeight<Parts>::OneConnectWeight(const Parts& initp, const OneConnect
Weight<Parts>& initwptr)
  :external_(1), nux_(initwptr.get_nux()), nuy_(initwptr.get_nuy()), connect_(&initp)
, weight_(initwptr.get_weightptr())
{
}
template <class Parts>
inline OneConnectWeight<Parts>::OneConnectWeight(const OneConnectWeight<Parts>& x, co
nst Parts& initp)
  :external_(x.external_), nux_(x.nux_), nuy_(x.nuy_), connect_(&initp)
{
  if( x.external_ == 0 ){
    weight_ = new n_vector<double>;
    (*weight_) = *x.weight_;
  }
  else weight_ = x.weight_;
}

template <class Parts>
inline OneConnectWeight<Parts>::~OneConnectWeight(void)
{
  if( external_ == 0 ) delete weight_;
}

template <class Parts>
inline OneConnectWeight<Parts>::OneConnectWeight(const OneConnectWeight& x)
  :external_(x.external_), nux_(x.nux_), nuy_(x.nuy_), connect_(x.connect_)
{
  if( x.external_ == 0 ){
    weight_ = new n_vector<double>;
    (*weight_) = *x.weight_;
  }
  else weight_ = x.weight_;
}

template <class Parts>
inline OneConnectWeight<Parts>& OneConnectWeight<Parts>::operator=(const OneConnectWe
ight& x)
{
  if( &x != this ){
    external = x.external;
    connect_ = x.connect_;

    if( x.external_ == 0 ){
      delete weight_;
      weight_ = new n_vector<double>;
      *weight = *x.weight;
    }
    else weight_ = x.weight_;
  }
  return *this;
}

// methods for weight ----------------------------
template <class Parts>
inline int OneConnectWeight<Parts>::get_numweight(void) const
{
```

```
  return (*weight_).get_number();
}

template <class Parts>
inline const Parts* OneConnectWeight<Parts>::get_connectptr(void) const
{
  return connect_;
}

template <class Parts>
inline n_vector<double>* OneConnectWeight<Parts>::get_weightptr(void) const
{
  return weight_;
}

template <class Parts>
inline double OneConnectWeight<Parts>::get_weight(int n) const
{
  return (*weight_).get_element(n);
}

template <class Parts>
inline void OneConnectWeight<Parts>::add_weight(double inputw, int nux, int nuy)
{
  for(int m=0; m<nux; m++){
    for(int n=0; n<nuy; n++){
      (*weight_).add_element(inputw);
    }
  }
}

template <class Parts>
inline void OneConnectWeight<Parts>::add_weight(int nux, int nuy, double(*initwf)(dou
ble x, double y))
{
  for(int m=0; m<nux; m++){
    for(int n=0; n<nuy; n++){
      (*weight_).add_element(initwf(m,n));
    }
  }
}

template <class Parts>
inline void OneConnectWeight<Parts>::rm_weight(int n)
{
  (*weight_).rm_element(n);
}

template <class Parts>
inline void OneConnectWeight<Parts>::set_weight(int n, double inputw)
{
  (*weight_).set_element(n, inputw);
}

template <class Parts>
inline void OneConnectWeight<Parts>::delta_weight(int n, double deltaw)
{
  (*weight_).delta_element(n, deltaw);
}

template <class Parts>
inline void OneConnectWeight<Parts>::show_weight(void)
{
  for(int i=0; i<(*weight_).get_number(); i++){
    cout << "Weight No. " << i << " : " << (*weight_).get_element(i) << "\n";
  }cout << "\n";
```



```
}

template <class Parts>
inline int OneConnectWeight<Parts>::get_nux(void) const
{
  return nux_;
}

template <class Parts>
inline int OneConnectWeight<Parts>::get_nuy(void) const
{
  return nuy_;
}

#endif // ___ONECONNECTWEIGHT_H___
```

```
// -*-c++-*-
// Neuron ConnectWeight Class

#ifndef ___CONNECTWEIGHT_H___
#define ___CONNECTWEIGHT_H___

#include "OneConnectWeight.h"

template <class Parts>
class ConnectWeight
{
private:
  int numconnect_;
  vector< OneConnectWeight<Parts>* >* connect_;

public:
  ConnectWeight(void);
  virtual ~ConnectWeight(void);
  ConnectWeight(const ConnectWeight& x);
  ConnectWeight& operator=(const ConnectWeight& x);

  // methods for connection
  virtual int find_connect(const Parts& parts) const;
  virtual int get_numconnect(void) const;

  virtual int get_numweight(int n) const;
  virtual int get_numweight(const Parts& t_parts) const;
  virtual const Parts* get_connectptr(int n) const;
  virtual double get_weight(int m, int n =0) const;
  virtual double get_weight(const Parts& parts, int n =0) const;

  virtual void show_connect(void) const;
  virtual void show_weight(void) const;
  virtual void show_connectweight(void) const;
  virtual void mk_connect(const Parts& newparts, double initw =0.0, int nux =1, int n
uy =1);
  virtual void mk_connect(const Parts& newparts, int nux, int nuy, double(*initwf)(do
uble x, double y));
  virtual void mk_connect(const Parts& newparts, const Parts& baseparts);
  virtual void rm_connect(const Parts& rmparts);
  virtual void ch_connect(const Parts& oldparts, const Parts& newparts);

  // methods for weight
  virtual void add_weight(const Parts& t_parts, double input);
  virtual void rm_weight(const Parts& t_parts, int n);
  virtual void set_weight(const Parts& t_parts, double inputw, int n =0);
  virtual void delta_weight(const Parts& t_parts, double deltaw, int n =0);

  virtual int get_nux(const Parts& t_parts) const;
  virtual int get_nuy(const Parts& t_parts) const;
};

// methods for ConnectWeight
template <class Parts>
inline ConnectWeight<Parts>::ConnectWeight(void)
  :numconnect_(0)
{
  connect_ = new vector< OneConnectWeight<Parts>* >;
  (*connect_).clear();
}

template <class Parts>
inline ConnectWeight<Parts>::~ConnectWeight(void)
{
  for(int i=0; i<numconnect_; i++) delete (*connect_)[i];
  delete connect_;
```



```
}

template <class Parts>
inline ConnectWeight<Parts>::ConnectWeight(const ConnectWeight& x)
  :numconnect_(x.numconnect_), connect_(new vector< OneConnectWeight<Parts>* >(*x.con
nect_))
{
}

template <class Parts>
inline ConnectWeight<Parts>& ConnectWeight<Parts>::operator=(const ConnectWeight& x)
{
  if( &x != this ){
    for(int i=0; i<numconnect_; i++) delete (*connect_)[i];
    delete connect_;

    numconnect_ = x.numconnect_;
    connect_ = new vector< OneConnectWeight<Parts>* >(*x.connect_);
  }
  return *this;
}

// methods for connection ------------------------------
template <class Parts>
inline int ConnectWeight<Parts>::find_connect(const Parts& parts) const
{
  for(int i=0; i<numconnect_; i++){
    if( (*connect_)[i]->get_connectptr() == &parts) return i;
  }
  return -1;
}

template <class Parts>
inline int ConnectWeight<Parts>::get_numconnect(void) const
{
  return numconnect_;
}

template <class Parts>
inline int ConnectWeight<Parts>::get_numweight(int n) const
{
  return (*connect_)[n]->get_numweight();
}

template <class Parts>
inline int ConnectWeight<Parts>::get_numweight(const Parts& t_parts) const
{
  int i = find_connect(t_parts);
  return (*connect_)[i]->get_numweight();
}

template <class Parts>
inline const Parts* ConnectWeight<Parts>::get_connectptr(int n) const
{
  return (*connect_)[n]->get_connectptr();
}

template <class Parts>
inline double ConnectWeight<Parts>::get_weight(int m, int n) const
{
  return (*connect_)[m]->get_weight(n);
}

template <class Parts>
inline double ConnectWeight<Parts>::get_weight(const Parts& parts, int n) const
{
```

```
  int i = find_connect(parts);
  if( !(i < 0) ){
    return (*connect_)[i]->get_weight(n);
  }
  else return 0.0;
}

template <class Parts>
inline void ConnectWeight<Parts>::show_connect(void) const
{
  int nux, nuy;

  for(int i=0; i<numconnect_; i++){
    cout << "Connection No. " << i << " :\n";
    nux = (*connect_)[i]->get_nux();
    nuy = (*connect_)[i]->get_nuy();
    for(int m=0; m<nux; m++){
      for(int n=0; n<nuy; n++){
        cout << "connect " << (*connect_)[i]->get_connectptr() << ", ";
      }cout << "\n";
    }
  }cout << "\n";
}

template <class Parts>
inline void ConnectWeight<Parts>::show_weight(void) const
{
  int nux, nuy;

  for(int i=0; i<numconnect_; i++){
    cout << "Connection No. " << i << " :\n";
    nux = (*connect_)[i]->get_nux();
    nuy = (*connect_)[i]->get_nuy();
    for(int m=0; m<nux; m++){
      for(int n=0; n<nuy; n++){
        cout << "weight " << (*connect_)[i]->get_weight(m*nux+n) << ", ";
      }cout << "\n";
    }
  }cout << "\n";
}

template <class Parts>
inline void ConnectWeight<Parts>::show_connectweight(void) const
{
  int nux, nuy;

  for(int i=0; i<numconnect_; i++){
    cout << "Connection No." << i << " :\n";
    nux = (*connect_)[i]->get_nux();
    nuy = (*connect_)[i]->get_nuy();
    for(int m=0; m<nux; m++){
      for(int n=0; n<nuy; n++){
        cout << "Connect to pointer " << (*connect_)[i]->get_connectptr() << ", "
             << "Weight" << m*nux+n << "  " << (*connect_)[i]->get_weight(m*nux+n) <<
 "\n";
      }cout << "\n";
    }
  }cout << "\n";
}

template <class Parts>
inline void ConnectWeight<Parts>::mk_connect(const Parts& newparts, double initw, int
 nux, int nuy)
{
  int i = find_connect(newparts);
  if( i < 0 ){
```



```
    OneConnectWeight<Parts>* Temp = new OneConnectWeight<Parts>(newparts, initw, nux,
 nuy);
    numconnect_++;
    (*connect_).push_back(Temp);
  }
  else{
    (*connect_)[i]->add_weight(initw, nux, nuy);
  }
}

template <class Parts>
inline void ConnectWeight<Parts>::mk_connect(const Parts& newparts, int nux, int nuy,
 double(*initwf)(double x, double y))
{
  int i = find_connect(newparts);
  if( i < 0 ){
    OneConnectWeight<Parts>* Temp = new OneConnectWeight<Parts>(newparts, nux, nuy, i
nitwf);
    numconnect_++;
    (*connect_).push_back(Temp);
  }
  else{
    (*connect_)[i]->add_weight(nux, nuy, initwf);
  }
}

template <class Parts>
inline void ConnectWeight<Parts>::mk_connect(const Parts& newparts, const Parts& base
parts)
{
  int i = find_connect(newparts);
  if( i < 0 ){

    int i = find_connect(baseparts);
    if( !(i < 0) ){
      OneConnectWeight<Parts>* Temp = new OneConnectWeight<Parts>(newparts, *(*connec
t_)[i]);

      numconnect_++;
      (*connect_).push_back(Temp);
    }
  }
}

template <class Parts>
inline void ConnectWeight<Parts>::rm_connect(const Parts& rmparts)
{
  int i = find_connect(rmparts);
  if( !(i < 0) ){
    delete (*connect_)[i];
    (*connect_).erase((*connect_).begin()+i);
    numconnect_--;
  }
}

template <class Parts>
inline void ConnectWeight<Parts>::ch_connect(const Parts& oldparts, const Parts& newp
arts)
{
  int i = find_connect(oldparts);
  if( !(i < 0) ){
    OneConnectWeight<Parts>* Temp = new OneConnectWeight<Parts>(*(*connect_)[i], newp
arts);
    delete (*connect_)[i];
    (*connect_)[i] = Temp;
  }
```

```
}

// methods for weight -----------------------------
template <class Parts>
inline void ConnectWeight<Parts>::add_weight(const Parts& t_parts, double input)
{
  int i = find_connect(t_parts);
  if( !(i < 0) ) (*connect_)[i]->add_weight(input);
}

template <class Parts>
inline void ConnectWeight<Parts>::rm_weight(const Parts& t_parts, int n)
{
  int i = find_connect(t_parts);
  if( !(i < 0) ) (*connect_)[i]->rm_weight(n);
}


template <class Parts>
inline void ConnectWeight<Parts>::set_weight(const Parts& t_parts, double inputw, int
 n)
{
  int i = find_connect(t_parts);
  if( !(i < 0) ) (*connect_)[i]->set_weight(n, inputw);
}

template <class Parts>
inline void ConnectWeight<Parts>::delta_weight(const Parts& t_parts, double deltaw, i
nt n)
{
  int i = find_connect(t_parts);
  if( !(i < 0) ) (*connect_)[i]->delta_weight(n, deltaw);
}

template <class Parts>
inline int ConnectWeight<Parts>::get_nux(const Parts& t_parts) const
{
  int i = find_connect(t_parts);
  if( !(i < 0) ){
    return (*connect_)[i]->get_nux();
  }
  else return 0;
}

template <class Parts>
inline int ConnectWeight<Parts>::get_nuy(const Parts& t_parts) const
{
  int i = find_connect(t_parts);
  if( !(i < 0) ){
    return (*connect_)[i]->get_nuy();
  }
  else return 0;
}

#endif // ___CONNECTWEIGHT_H___
```

```
// -*-c++-*-
// Neuron Cell Class

#ifndef ___CELL_H___
#define ___CELL_H___

#include "ConnectWeight.h"

class Cell
{
private:
  double celloutput_;
  ConnectWeight<Cell> connectw_;

public:
  Cell(double initcell =0.0);
  virtual ~Cell(void);
  Cell(const Cell& x);
  Cell& operator=(const Cell& x);

  // methods for class ConnectWeight
  virtual int find_connect(const Cell& cell) const;
  virtual int get_numconnect(void) const;
  virtual double get_numweight(int n) const;
  virtual double get_numweight(const Cell& cell) const;
  virtual const Cell* get_connectptr(int n) const;
  virtual double get_weight(int m, int n =0) const;
  virtual double get_weight(const Cell& cell, int n =0) const;
  virtual void show_connect(void) const;
  virtual void show_weight(void) const;
  virtual void show_connectweight(void) const;
  virtual void mk_connect(const Cell& newcell, double initw =0.0);
  virtual void mk_connect(const Cell& newcell, const Cell& basecell);
  virtual void rm_connect(const Cell& rmcell);
  virtual void ch_connect(const Cell& oldcell, const Cell& newcell);
  virtual void set_weight(const Cell& t_cell, double inputw, int n =0);
  virtual void delta_weight(const Cell& t_cell, double deltaw, int n =0);

  // methods for class Cell
  virtual double allconnect_out(void) const;
  virtual double partconnect_out(int n) const;
  virtual double partconnect_out(const Cell& t_cell) const;

  virtual double get_celloutput(void) const;
  virtual void setforce_celloutput(double input);

  // output functions
  virtual double output_allconnect(void);
  virtual double output_partconnect(const Cell& t_cell);
  virtual double output_sigmoid(double th =0.0, double coef =1.0);
};

#endif // ___CELL_H___
```

```
#include <iostream>
#include <cmath>
#include "Cell.h"

using namespace std;

Cell::Cell(double initcell)
  :celloutput_(initcell)
{
}

Cell::~Cell(void)
{
}

Cell::Cell(const Cell& x)
  : celloutput_(x.celloutput_), connectw_(x.connectw_)
{
}

Cell&
Cell::operator=(const Cell& x)
{
  if(&x != this){
    celloutput_ = x.celloutput_;
    connectw_ = x.connectw_;
  }
  return *this;
}

// methods for class ConnectWeight ------------------------
int
Cell::find_connect(const Cell& cell) const
{
  return connectw_.find_connect(cell);
}

int
Cell::get_numconnect(void) const
{
  return connectw_.get_numconnect();
}

double
Cell::get_numweight(int n) const
{
  return connectw_.get_numweight(n);
}

double
Cell::get_numweight(const Cell& cell) const
{
  return connectw_.get_numweight(cell);
}

const Cell*
Cell::get_connectptr(int n) const
{
  return connectw_.get_connectptr(n);
}

double
Cell::get_weight(int m, int n) const
{
  return connectw_.get_weight(m, n);
}
```



```
double
Cell::get_weight(const Cell& cell, int n) const
{
  return connectw_.get_weight(cell, n);
}

void
Cell::show_connect(void) const
{
  connectw_.show_connect();
}

void
Cell::show_weight(void) const
{
  connectw_.show_weight();
}

void
Cell::show_connectweight(void) const
{
  connectw_.show_connectweight();
}

void
Cell::mk_connect(const Cell& newcell, double initw)
{
  connectw_.mk_connect(newcell, initw);
}

void
Cell::mk_connect(const Cell& newcell, const Cell& basecell)
{
  connectw_.mk_connect(newcell, basecell);
}

void
Cell::rm_connect(const Cell& rmcell)
{
  connectw_.rm_connect(rmcell);
}

void
Cell::ch_connect(const Cell& oldcell, const Cell& newcell)
{
  connectw_.ch_connect(oldcell, newcell);
}

void
Cell::set_weight(const Cell& t_cell, double inputw, int n)
{
  connectw_.set_weight(t_cell, inputw, n);
}

void
Cell::delta_weight(const Cell& t_cell, double deltaw, int n)
{
  connectw_.delta_weight(t_cell, deltaw, n);
}

double
Cell::allconnect_out(void) const
{
  double temp =0.0;
```

```
  for(int i=0; i<connectw_.get_numconnect(); i++){
    for(int j=0; j<connectw_.get_numweight(i); j++){
      temp += (connectw_.get_connectptr(i)->get_celloutput())*(connectw_.get_weight(i
, j));
    }
  }

  return temp;
}

double
Cell::partconnect_out(int n) const
{
  double temp =0.0;

  for(int j=0; j<connectw_.get_numweight(n); j++){
    temp += connectw_.get_connectptr(n)->get_celloutput()*connectw_.get_weight(n, j);
  }

  return temp;
}

double
Cell::partconnect_out(const Cell& t_cell) const
{
  double temp =0.0;

  int i = connectw_.find_connect(t_cell);
  for(int j=0; j<connectw_.get_numweight(i); j++){
    temp += connectw_.get_connectptr(i)->get_celloutput()*connectw_.get_weight(i, j);
  }

  return temp;
}

// methods for class Cell -------------------------
double
Cell::get_celloutput(void) const
{
  return celloutput_;
}

void
Cell::setforce_celloutput(double inputcell)
{
  celloutput_ = inputcell;
}

// output functions
double
Cell::output_allconnect(void)
{
  double temp =0.0;

  for(int i=0; i<connectw_.get_numconnect(); i++){
    for(int j=0; j<connectw_.get_numweight(i); j++){
      temp += (connectw_.get_connectptr(i)->get_celloutput())*(connectw_.get_weight(i
, j));
    }
  }

  celloutput_ = temp;
  return celloutput_;
}

double
```



```
Cell::output_partconnect(const Cell& t_cell)
{
  double temp =0.0;

  int i = connectw_.find_connect(t_cell);
  for(int j=0; j<connectw_.get_numweight(i); j++){
    temp += connectw_.get_connectptr(i)->get_celloutput() * connectw_.get_weight(i, j
);
  }

  celloutput_ = temp;
  return celloutput_;
}

double
Cell::output_sigmoid(double th, double coef)
{
  double temp = allconnect_out();
  celloutput_ = 1.0/( 1.0 + exp( -coef*( temp-th ) ) );
  return celloutput_;
}
```

```
// -*-c++-*-
// Cell-Plane Class

#ifndef ___PLANE_H___
#define ___PLANE_H___

#include "ConnectWeight.h"

class Cell;

class Plane
{
private:
  int plane_x_, plane_y_;
  ConnectWeight<Plane> connectw_;

public:
  Cell** cell;

  // constructor
  Plane(int px, int py, double initcell =0.0);
  virtual ~Plane(void);
  Plane(const Plane& x);
  Plane& operator=(const Plane& x);

  // methods for class ConnectWeight
  virtual int find_connect(const Plane& plane) const;
  virtual int get_nux(const Plane& plane) const;
  virtual int get_nuy(const Plane& plane) const;
  virtual int get_numconnect(void) const;
  virtual double get_numweight(int n) const;
  virtual double get_numweight(const Plane& plane) const;
  virtual const Plane* get_connectptr(int n) const;
  virtual double get_weight(int m, int n =0) const;
  virtual double get_weight(const Plane& plane, int n =0) const;
  virtual void show_connect(void) const;
  virtual void mk_connect(const Plane& newplane, double initw, int nx, int ny);
  virtual void mk_connect(const Plane& newplane, int nx, int ny, double(*initwf)(doub
le x, double y));
  virtual void mk_connect(const Plane& newplane, const Plane& baseplane);
  virtual void rm_connect(const Plane& rmplane);
  virtual void ch_connect(const Plane& oldplane, const Plane& newplane);
  virtual void set_weight(const Plane& t_plane, double inputw, int n);
  virtual void delta_weight(const Plane& t_plane, double deltaw, int n);

  // methods for class Cell
  virtual double get_celloutput(int x, int y) const;
  virtual void setforce_celloutput(int x, int y, double input);

  // methods for class Plane
  virtual int get_planex(void) const;
  virtual int get_planey(void) const;
  virtual Cell* get_cellptr(int x, int y) const;
  virtual void show_plane(void) const;
  virtual void show_cellptr(void) const;
  virtual void set_input(double** input);
  virtual void select_seedcell(int* x, int* y, double* celloutput) const;

  // output functions
  virtual void output_subSlayer(const Plane& t_plane);

  //for test
  virtual void print_pgm(char* filename) const;
  virtual void print_data(char* filename) const;
};
```



```
#endif //___PLANE_H___
```

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include "Cell.h"
#include "Plane.h"

using namespace std;

Plane::Plane(int px, int py, double initcell)
  :plane_x_(px), plane_y_(py)
{
  cell = new Cell*[plane_x_];
  for(int i=0; i<plane_x_; i++){
    cell[i] = new Cell[plane_y_](initcell);
  }
}

Plane::~Plane(void)
{
  for(int i=0; i<plane_y_; i++){
    delete[] cell[i];
  }
  delete[] cell;
}

Plane::Plane(const Plane& x)
  :plane_x_(x.plane_x_), plane_y_(x.plane_y_), connectw_(x.connectw_)
{
  cell = new Cell*[x.plane_x_];
  for(int i=0; i<x.plane_x_; i++){
    cell[i] = new Cell[x.plane_y_];
    for(int j=0; j<x.plane_y_; j++){
      cell[i][j] = x.cell[i][j];
    }
  }
}

Plane&
Plane::operator=(const Plane& x)
{
  if( &x != this ){

    for(int i=0; i<plane_y_; i++){
      delete[] cell[i];
    }
    delete[] cell;

    plane_x_ = x.plane_x_;
    plane_y_ = x.plane_y_;

    connectw_ = x.connectw_;
    cell = new Cell*[x.plane_x_];
    for(int i=0; i<x.plane_x_; i++){
      cell[i] = new Cell[x.plane_y_];
      for(int j=0; j<x.plane_y_; j++){
        cell[i][j] = x.cell[i][j];
      }
    }
  }
  return *this;
}

// methods for class ConnectWeight -------------------------
int
Plane::find_connect(const Plane& plane) const
{
```



```
  return connectw_.find_connect(plane);
}

int
Plane::get_nux(const Plane& plane) const
{
  return connectw_.get_nux(plane);
}

int
Plane::get_nuy(const Plane& plane) const
{
  return connectw_.get_nuy(plane);
}

int
Plane::get_numconnect(void) const
{
  return connectw_.get_numconnect();
}

double
Plane::get_numweight(int n) const
{
  return connectw_.get_numweight(n);
}

double
Plane::get_numweight(const Plane& plane) const
{
  return connectw_.get_numweight(plane);
}

const Plane*
Plane::get_connectptr(int n) const
{
  return connectw_.get_connectptr(n);
}

double
Plane::get_weight(int m, int n) const
{
  return connectw_.get_weight(m, n);
}

double
Plane::get_weight(const Plane& plane, int n) const
{
  return connectw_.get_weight(plane, n);
}

void
Plane::show_connect(void) const
{
  int nux, nuy;

  for(int i=0; i<connectw_.get_numconnect(); i++){
    cout << "Connection No. " << i << " :\n";
    nux = connectw_.get_nux(*connectw_.get_connectptr(i));
    nuy = connectw_.get_nuy(*connectw_.get_connectptr(i));
    for(int m=0; m<nux; m++){
      for(int n=0; n<nuy; n++){
        cout << "weight " << connectw_.get_weight(i, m*nux+n) << ", ";
      }cout << "\n";
    }cout << "\n";
  }cout << "\n";
```

```
}

void
Plane::mk_connect(const Plane& newplane, double initw, int nux, int nuy)
{
  connectw_.mk_connect(newplane, initw, nux, nuy);
}

void
Plane::mk_connect(const Plane& newplane, int nux, int nuy, double(*initwf)(double x,
double y))
{
  connectw_.mk_connect(newplane, nux, nuy, initwf);
}

void
Plane::mk_connect(const Plane& newplane, const Plane& baseplane)
{
  connectw_.mk_connect(newplane, baseplane);
}

void
Plane::rm_connect(const Plane& rmplane)
{
  connectw_.rm_connect(rmplane);
}

void
Plane::ch_connect(const Plane& oldplane, const Plane& newplane)
{
  connectw_.ch_connect(oldplane, newplane);
}

void
Plane::set_weight(const Plane& t_plane, double inputw, int n)
{
  connectw_.set_weight(t_plane, inputw, n);
}

void
Plane::delta_weight(const Plane& t_plane, double deltaw, int n)
{
  connectw_.delta_weight(t_plane, deltaw, n);
}

// methods for class Cell
double
Plane::get_celloutput(int x, int y) const
{
  if( x >= 0 && plane_x_  > x && y >= 0 && plane_y_ > y ) return cell[x][y].get_cello
utput();
  else return 0.0;
}

void
Plane::setforce_celloutput(int x, int y, double input)
{
  if( x >= 0 && plane_x_  > x && y >= 0 && plane_y_ > y ) cell[x][y].setforce_cellout
put(input);
}

// methods for class Plane -------------------------
int
Plane::get_planex(void) const
{
  return plane_x_;
```



```
}

int
Plane::get_planey(void) const
{
  return plane_y_;
}

Cell*
Plane::get_cellptr(int x, int y) const
{
  if( x >= 0 && plane_x_  > x && y >= 0 && plane_y_ > y ) return &cell[x][y];
  else return NULL;
}

void
Plane::show_plane(void) const
{
  for(int i=0; i<plane_x_; i++){
    for(int j=0; j<plane_y_; j++){
      cout << setw(8) << cell[i][j].get_celloutput() << " ";
    }
    cout << endl;
  }
  cout << endl;
}

void
Plane::show_cellptr(void) const
{
  for(int i=0; i<plane_x_; i++){
    for(int j=0; j<plane_y_; j++){
      cout << &cell[i][j] << " ";
    }cout << endl;
  }cout << endl;
}

void
Plane::set_input(double** input)
{
  for(int i=0; i<plane_x_; i++){
    for(int j=0; j<plane_y_; j++){
      cell[i][j].setforce_celloutput(input[i][j]);
    }
  }
}

void
Plane::select_seedcell(int* x, int* y, double* celloutput) const
{
  *celloutput = 0.0;

  for(int i=0; i<get_planex(); i++){
    for(int j=0; j<get_planey(); j++){
      if( cell[i][j].get_celloutput() > *celloutput ){
        *x = i;
        *y = j;
        *celloutput = cell[i][j].get_celloutput();
      }
    }
  }
}

void
Plane::output_subSlayer(const Plane& t_plane)
{
```

```
  double dtemp =0.0;

  int nux = get_nux(t_plane);
  int nuy = get_nuy(t_plane);

  int shiftx = (t_plane.get_planex() - plane_x_ -nux +1)/2;
  int shifty = (t_plane.get_planey() - plane_y_ -nuy +1)/2;

  for(int i=0; i<plane_x_; i++){
    for(int j=0; j<plane_y_; j++){

      for(int m=0; m<nux; m++){
        for(int n=0; n<nuy; n++){

          dtemp += t_plane.cell[i+m+shiftx][j+n+shifty].get_celloutput() * get_weight
(t_plane, m*nux+n);

        }
      }

      cell[i][j].setforce_celloutput(dtemp);
      dtemp = 0.0;
    }
  }
}

void
Plane::print_pgm(char* filename) const
{
  ofstream fout;
  double max =0.0;

  fout.open(filename);
  if(fout.fail()){
    cerr << "cannot open output file" << endl;
    exit(0);
  }

  for(int i=0; i<get_planex(); i++){
    for(int j=0; j<get_planey(); j++){
      if( cell[i][j].get_celloutput() > max )
        max = cell[i][j].get_celloutput();
    }
  }

  fout << "P2 " << get_planex() << " " << get_planey() << " 255\n";
  for(int i=0; i<get_planex(); i++){
    for(int j=0; j<get_planey(); j++){
      if( max == 0.0 ) fout << setw(3) << 255 << " ";
      else fout << setw(3) << (int)(255.0 - cell[i][j].get_celloutput()/max*255.0) <<
 " ";
    }fout << "\n";
  }
  fout.close();
}

void
Plane::print_data(char* filename) const
{
  ofstream fout;

  fout.open(filename);
  if(fout.fail()){
    cerr << "cannot open output file" << endl;
    exit(0);
  }
```



```
  for(int i=0; i<get_planex(); i++){
    for(int j=0; j<get_planey(); j++){
      fout << setw(3) << i << " " << setw(3) << j << setw(10) << cell[i][j].get_cello
utput() << "\n";
    }fout << "\n";
  }
  fout.close();
}
```

```
// -*-c++-*-
// Layer Class

#ifndef ___LAYER_H___
#define ___LAYER_H___

#include <vector>
#include "ConnectWeight.h"
#include "Commonf.h"

class Cell;
class Plane;

class Layer
{
private:
  const int defaultpx_, defaultpy_; //default plane sizes
  const double defaultinitcell_; //default cell output
  int numplane_;
  ConnectWeight<Layer> connectw_;

public:
  vector<Plane*> plane;

  // constructor
  Layer(int nump, int px, int py, double initcell =0.0);
  virtual ~Layer(void);
  Layer(const Layer& x);
  Layer& operator=(const Layer& x);

  // methods for class ConnectWeight
  virtual int find_connect(const Layer& layer) const;
  virtual int get_numconnect(void) const;
  virtual double get_numweight(int n) const;
  virtual double get_numweight(const Layer& layer) const;
  virtual const Layer* get_connectptr(int n) const;
  virtual double get_weight(int m, int n =0) const;
  virtual double get_weight(const Layer& layer, int n =0) const;
  virtual void show_connect(void) const;
  virtual void mk_connect(const Layer& newlayer, double initw =0.0);
  virtual void mk_connect(const Layer& newlayer, const Layer& baselayer);
  virtual void rm_connect(const Layer& rmlayer);
  virtual void ch_connect(const Layer& oldlayer, const Layer& newlayer);
  virtual void set_weight(const Layer& t_layer, double inputw, int n =0);
  virtual void delta_weight(const Layer& t_layer, double deltaw, int n =0);

  // methods for Planes
  virtual void show_connect_plane(void) const;

  // methods for Layers
  virtual int get_numplane(void) const;
  virtual Plane* get_planeptr(int nump) const;
  virtual void show_layer(void) const;

  virtual void mk_fullconnect(const Layer& t_ps, double initw, int nux, int nuy);
  virtual void mk_1to1connect(const Layer& t_ps, double initw, int nux, int nuy);

  virtual void mk_fullcommon(const Layer& t_ps, double initw, int nux, int nuy);
  virtual void mk_1to1common(const Layer& t_ps, double initw, int nux, int nuy);

  virtual void mk_fullconnect_gauss(const Layer& t_ps, int nux, int nuy, double sigma
x =1.0, double sigmay =1.0);
  virtual void mk_1to1connect_gauss(const Layer& t_ps, int nux, int nuy, double sigma
x =1.0, double sigmay =1.0);

  virtual void mk_fullconnect_f(const Layer& t_ps, int nux, int nuy, double(*initwf)(
```



```
double x, double y));
  virtual void mk_1to1connect_f(const Layer& t_ps, int nux, int nuy, double(*initwf)(
double x, double y), int(*convergef)(int n) =straight);

  virtual void mk_Sconnect(const Layer& clayer, double initwc, const Layer& vlayer, i
nt numvp, double initwv, int nux, int nuy);
  virtual void mk_stdSconnect(const Layer& clayer, double initwc, Layer& vlayer, int
numvp, double initwv, int nux, int nuy, double(*initwf)(double x, double y) =initw1);
  virtual void mk_onestdSconnect(int numsp, const Layer& clayer, double initwc, const
 Layer& vlayer, int numvp, double initwv, int nux, int nuy);
  //make connections automatically. clayer connects nux*nuy, vlayer connects 1*1.

  virtual void mk_stdCconnect(const Layer& slayer, int nux, int nuy, double(*initwf)(
double x, double y) =initw1, int(*convergef)(int n) =straight);
  //make connections automatically. slayer connects nux*nuy,
  //under convergef condition, and initialize the weights initwf function.

  virtual void mk_stdVconnect(const Layer& clayer, int nux, int nuy, double(*initwf)(
double x, double y) =initw1);
  virtual void mk_onestdVconnect(int numvp, const Layer& clayer, int nux, int nuy, do
uble(*initwf)(double x, double y) =initw1);
  //make connections automatically. clayer connects nux*nuy,
  //and initialize the weights initwf function.

  virtual int add_plane(void);
  virtual void del_plane(int num_p);

  //output functions
  virtual void output_Slayer(const Layer& clayer  const Layer& vlayer, int numvp, dou
ble r, double sigma =1.0); // only slayer output

  virtual void output_stdSlayer(const Layer& clayer, Layer& vlayer, int numvp, double
 r, double sigma =1.0); //slyaer and vlayer output

  virtual void output_stdClayer(const Layer& slayer);

  virtual void output_stdVlayer(int nump, const Layer& clayer);

  //for learning
  virtual void response_stdSlayer(int x, int y, const Layer& clayer, Layer& vlayer, i
nt numvp, double r, double sigma =1.0);
  //one cell are calculated to get its response of firing

  virtual Plane* mk_ssp(const Layer& clayer, int nux, int nuy) const;
  virtual int find_max_firing_plane(int x, int y) const;

  virtual void learning_excite(int nump, const Layer& prvclayer, int cx_index, int cy
_index, double q, double(*weightc)(double x, double y));
  virtual void learning_inhibit(int nump, const Layer& vlayer, int numvp, int vx_inde
x, int vy_index, double q);

  //for test
  virtual void print_pgm(char* name) const;
  virtual void print_data(char* name) const;
  virtual void print_pgm(char* name, int index1, int index2 =0, int index3 =0) const;
  virtual void print_data(char* name, int index1, int index2 =0, int index3 =0) const
;
  virtual void print_pgm_data(char* name, int index1, int index2 =0, int index3 =0) c
onst;
  virtual void printall_pgm(char* name, int numx, int index1 =0, int index2 =0) const
;
  virtual void printall_data(char* name, int index1 =0, int index2 =0) const;
  virtual void printall_pgm_data(char* name, int numx, int index1 =0, int index2 =0)
const;

};
```

```
#endif //___LAYER_H___
```

```
#include <iostream>
#include <strstream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include "Cell.h"
#include "Plane.h"
#include "Layer.h"

using namespace std;

Layer::Layer(int nump, int px, int py, double initcell)
  :defaultpx_(px), defaultpy_(py), defaultinitcell_(initcell), numplane_(0)
{
  plane.clear();

  for(int i=0; i<nump; i++){
    add_plane();
  }
}

Layer::~Layer(void)
{
  for(int i=0; i<numplane_; i++){
    delete plane[i];
  }
}

Layer::Layer(const Layer& x)
  :defaultpx_(x.defaultpx_), defaultpy_(x.defaultpy_), defaultinitcell_(x.defaultinit
cell_), numplane_(0), connectw_(x.connectw_)
{
  for(int i=0; i<x.numplane_; i++){
    plane[i] = x.plane[i];
  }
}

Layer&
Layer::operator=(const Layer& x)
{
  if( &x != this ){
    for(int i=0; i<numplane_; i++){
      delete plane[i];
    }
    plane.clear();

    numplane_ = x.numplane_;
    connectw_ = x.connectw_;
    for(int i=0; i<x.numplane_; i++){
      plane[i] = x.plane[i];
    }
  }
  return *this;
}

// methods for class ConnectWeight -------------------------
int
Layer::find_connect(const Layer& layer) const
{
  return connectw_.find_connect(layer);
}

int
Layer::get_numconnect(void) const
{
  return connectw_.get_numconnect();
```



```
}

double
Layer::get_numweight(int n) const
{
  return connectw_.get_numweight(n);
}

double
Layer::get_numweight(const Layer& layer) const
{
  return connectw_.get_numweight(layer);
}

const Layer*
Layer::get_connectptr(int n) const
{
  return connectw_.get_connectptr(n);
}

double
Layer::get_weight(int m, int n) const
{
  return connectw_.get_weight(m, n);
}

double
Layer::get_weight(const Layer& layer, int n) const
{
  return connectw_.get_weight(layer, n);
}

void
Layer::show_connect(void) const
{
  connectw_.show_connect();
}

void
Layer::mk_connect(const Layer& newlayer, double initw)
{
  connectw_.mk_connect(newlayer, initw);
}

void
Layer::mk_connect(const Layer& newlayer, const Layer& baselayer)
{
  connectw_.mk_connect(newlayer, baselayer);
}

void
Layer::rm_connect(const Layer& rmlayer)
{
  connectw_.rm_connect(rmlayer);
}

void
Layer::ch_connect(const Layer& oldlayer, const Layer& newlayer)
{
  connectw_.ch_connect(oldlayer, newlayer);
}

void
Layer::set_weight(const Layer& t_layer, double inputw, int n)
{
  connectw_.set_weight(t_layer, inputw, n);
```

```
}

void
Layer::delta_weight(const Layer& t_layer, double deltaw, int n)
{
  connectw_.delta_weight(t_layer, deltaw, n);
}

// methods for Planes ----------------------
void
Layer::show_connect_plane(void) const
{
  for(int i=0 ; i<numplane_; i++){
    cout << "Plane No. " << i << "\n";
    plane[i]->show_connect();
  }cout << "\n";
}

// methods for Layers ----------------------
int
Layer::get_numplane(void) const
{
  return numplane_;
}

Plane*
Layer::get_planeptr(int nump) const
{
  return plane[nump];
}

void
Layer::show_layer(void) const
{
  cout << "Plane in Layer" <<"\n";
  for(int i=0; i<numplane_; i++){
    cout << "Plane No. " << i << "\n";
    plane[i]->show_plane();
  }
}

void
Layer::mk_fullconnect(const Layer& t_ps, double initw, int nux, int nuy)
{
  for(int i=0; i<numplane_; i++){
    for(int j=0; j<t_ps.get_numplane(); j++){
      plane[i]->mk_connect(*t_ps.get_planeptr(j), initw, nux, nuy);
    }
  }
}

void
Layer::mk_1to1connect(const Layer& t_ps, double initw, int nux, int nuy)
{
  if( numplane_ > t_ps.get_numplane() )
    cerr << "out of range : 1to1 connection" << "\n";

  for(int i=0; i<numplane_; i++){
    plane[i]->mk_connect(*t_ps.get_planeptr(i), initw, nux, nuy);
  }
}

void
Layer::mk_fullcommon(const Layer& t_ps, double initw, int nux, int nuy)
{
  for(int i=0; i<numplane_; i++){
```



```
    for(int j=0; j<t_ps.get_numplane(); j++){
      if( j == 0 ) plane[i]->mk_connect(*t_ps.get_planeptr(j), initw, nux, nuy);
      else plane[i]->mk_connect(*t_ps.get_planeptr(j), *t_ps.get_planeptr(0));
    }
  }
}

void
Layer::mk_1to1common(const Layer& t_ps, double initw, int nux, int nuy)
{
  for(int i=0; i<numplane_; i++){
    if( i == 0 ) plane[i]->mk_connect(*t_ps.get_planeptr(i), initw, nux, nuy);
    else plane[i]->mk_connect(*t_ps.get_planeptr(i), *t_ps.get_planeptr(0));
  }
}

void
Layer::mk_fullconnect_gauss(const Layer& t_ps, int nux, int nuy, double sigmax, doubl
e sigmay)
{
  double temp;

  for(int i=0; i<numplane_; i++){
    for(int j=0; j<t_ps.get_numplane(); j++){
      plane[i]->mk_connect(*t_ps.get_planeptr(j), 0.0, nux, nuy);

      for(int m=0; m<nux; m++){
        for(int n=0; n<nuy; n++){
          temp = fixedwf(m, n, t_ps.get_numplane(), nux, nuy, sigmax, sigmay);
          plane[i]->set_weight(*t_ps.get_planeptr(j), temp, m*nux+n);
        }
      }

    }
  }
}

void
Layer::mk_1to1connect_gauss(const Layer& t_ps, int nux, int nuy, double sigmax, doubl
e sigmay)
{
  double temp;

  for(int i=0; i<numplane_; i++){
    plane[i]->mk_connect(*t_ps.get_planeptr(i), 0.0, nux, nuy);
    for(int m=0; m<nux; m++){
      for(int n=0; n<nuy; n++){
        temp = fixedwf(m, n, t_ps.get_numplane(), nux, nuy, sigmax, sigmay);
        plane[i]->set_weight(*t_ps.get_planeptr(i), temp, m*nux+n);
      }
    }

  }
}

void
Layer::mk_fullconnect_f(const Layer& t_ps, int nux, int nuy, double(*initwf)(double x
, double y))
{
   for(int i=0; i<numplane_; i++){
    for(int j=0; j<t_ps.get_numplane(); j++){
      plane[i]->mk_connect(*t_ps.get_planeptr(j), nux, nuy, initwf);
    }
  }
}
```

```
void
Layer::mk_1to1connect_f(const Layer& t_ps, int nux, int nuy, double(*initwf)(double x
, double y), int(convergef)(int n))
{
  int temp;

  for(int i=0; i<t_ps.get_numplane(); i++){
    temp = convergef(i);
    plane[i]->mk_connect(*t_ps.get_planeptr(i), nux, nuy, initwf);
  }
}

void
Layer::mk_Sconnect(const Layer& clayer, double initwc, const Layer& vlayer, int numvp
, double initwv, int nux, int nuy)
{
  for(int i=0; i<numplane_; i++){
    for(int j=0; j<clayer.get_numplane(); j++){
      plane[i]->mk_connect(*clayer.get_planeptr(j), initwc, nux, nuy);
    }
  }

  for(int i=0; i<numplane_; i++){
    plane[i]->mk_connect(*vlayer.get_planeptr(numvp), initwv, 1, 1);
  }

}

void
Layer::mk_stdSconnect(const Layer& clayer, double initwc, Layer& vlayer, int numvp, d
ouble initwv, int nux, int nuy, double(*initwf)(double x, double y))
{
  vlayer.mk_stdVconnect(clayer, nux, nuy, initwf);
  mk_Sconnect(clayer, initwc, vlayer, numvp, initwv, nux, nuy);
}

void
Layer::mk_onestdSconnect(int numsp, const Layer& clayer, double initwc, const Layer&
vlayer, int numvp, double initwv, int nux, int nuy)
{
  for(int j=0; j<clayer.get_numplane(); j++){
    plane[numsp]->mk_connect(*clayer.get_planeptr(j), initwc, nux, nuy);
  }

  plane[numsp]->mk_connect(*vlayer.get_planeptr(numvp), initwv, 1, 1);
}

void
Layer::mk_stdCconnect(const Layer& slayer, int nux, int nuy, double(*initwf)(double x
, double y), int (*convergef)(int n))
{
  int i;

  for(int j=0; j<slayer.get_numplane(); j++){
    i = convergef(j);
    plane[i]->mk_connect(*slayer.get_planeptr(j), nux, nuy, initwf);
  }
}

void
Layer::mk_stdVconnect(const Layer& clayer, int nux, int nuy, double(*initwf)(double x
, double y))
{
   for(int i=0; i<numplane_; i++){
    for(int j=0; j<clayer.get_numplane(); j++){
      plane[i]->mk_connect(*clayer.get_planeptr(j), nux, nuy, initwf);
```



```
    }
  }
}

void
Layer::mk_onestdVconnect(int numvp, const Layer& clayer, int nux, int nuy, double(*in
itwf)(double x, double y))
{
  for(int j=0; j<clayer.get_numplane(); j++){
    plane[numvp]->mk_connect(*clayer.get_planeptr(j), nux, nuy, initwf);
  }
}

int
Layer::add_plane(void)
{
  Plane* ptrplane = new Plane(defaultpx_, defaultpy_, defaultinitcell_);

  plane.push_back(ptrplane);
  numplane_++;

  return numplane_-1; // index of new plane
}

void
Layer::del_plane(int nump)
{
  delete plane[nump];

  plane.erase(plane.begin()+nump);
  numplane_--;
}

void
Layer::output_Slayer(const Layer& clayer, const Layer& vlayer, int numvp, double r, d
ouble sigma)
{
  int index;
  int ctosnux, ctosnuy, vtosnux, vtosnuy;
  int shiftx, shifty;
  double dtemp1 =0.0;
  double dtemp2 =0.0;

  for(int k=0; k<numplane_; k++){
    for(int i=0; i<plane[k]->get_planex(); i++){
      for(int j=0; j<plane[k]->get_planey(); j++){

        // calculate numerator
        for(int ka=0; ka<clayer.get_numplane(); ka++){

          index = plane[k]->find_connect(*clayer.get_planeptr(ka));
          if( !(index < 0) ){

            ctosnux = plane[k]->get_nux(*clayer.get_planeptr(ka));
            ctosnuy = plane[k]->get_nuy(*clayer.get_planeptr(ka));

            shiftx = (clayer.plane[ka]->get_planex() - plane[k]->get_planex() - ctosn
ux +1)/2; // init x position
            shifty = (clayer.plane[ka]->get_planey() - plane[k]->get_planey() - ctosn
uy +1)/2; // init y position

            for(int m=0; m<ctosnux; m++){
              for(int n=0; n<ctosnuy; n++){
                dtemp1 += plane[k]->get_weight(index, m*ctosnux+n)*clayer.plane[ka]->
get_celloutput(i+m+shiftx, j+n+shifty);
              } // m
```

```
            } // n

          } //end if
        } // ka
        dtemp1 += sigma;
        // end of calculating numerator

        // calculate denominator
        index = plane[k]->find_connect(*vlayer.get_planeptr(numvp));

        vtosnux = plane[k]->get_nux(*vlayer.get_planeptr(numvp));
        vtosnuy = plane[k]->get_nuy(*vlayer.get_planeptr(numvp));

        for(int m=0; m<vtosnux; m++){
          for(int n=0; n<vtosnuy; n++){
            dtemp2 = sigma + r/(1.0 + r)*plane[k]->get_weight(index, 0)*vlayer.plane[
numvp]->cell[i][j].get_celloutput();
          }
        }
        // end of calculating denominator

        dtemp1 = r*phif(dtemp1/dtemp2 - 1.0);
        plane[k]->cell[i][j].setforce_celloutput(dtemp1);
        dtemp1 = 0.0;

      } // j
    } // i
  } // k

}

void
Layer::output_stdSlayer(const Layer& clayer, Layer& vlayer, int numvp, double r, doub
le sigma)
{
  vlayer.output_stdVlayer(numvp, clayer);
  output_Slayer(clayer, vlayer, numvp, r, sigma);
}

void
Layer::output_stdClayer(const Layer& slayer)
{
  int index;
  int nux, nuy;
  int shiftx, shifty;
  double dtemp1 =0.0;

  for(int k=0; k<numplane_; k++){
    for(int i=0; i<plane[k]->get_planex(); i++){
      for(int j=0; j<plane[k]->get_planey(); j++){

        for(int ka=0; ka<slayer.get_numplane(); ka++){
          index = plane[k]->find_connect(*slayer.get_planeptr(ka));

          if( !(index < 0) ){
            nux = plane[k]->get_nux(*slayer.get_planeptr(ka));
            nuy = plane[k]->get_nuy(*slayer.get_planeptr(ka));
            shiftx = (slayer.plane[ka]->get_planex() - plane[k]->get_planex() -nux +1
)/2; // init x position
            shifty = (slayer.plane[ka]->get_planey() - plane[k]->get_planey() -nuy +1
)/2; // init y position

            for(int m=0; m<nux; m++){
              for(int n=0; n<nuy; n++){
                dtemp1 += plane[k]->get_weight(index, m*nux+n)*slayer.plane[ka]->get_
celloutput(i+m+shiftx, j+n+shifty);
```



```
              }
            }

          } //end if
        } // ka
        dtemp1 = psif(dtemp1);
        plane[k]->cell[i][j].setforce_celloutput(dtemp1);
        dtemp1 = 0.0;
      } // j
    } // i
  } // k
}

void
Layer::output_stdVlayer(int numvp, const Layer& clayer)
{
  int index;
  int nux, nuy;
  int shiftx, shifty;
  double dtemp1 =0.0;

  for(int i=0; i<plane[numvp]->get_planex(); i++){
    for(int j=0; j<plane[numvp]->get_planey(); j++){

      for(int ka=0; ka<clayer.get_numplane(); ka++){
        index = plane[numvp]->find_connect(*clayer.get_planeptr(ka));

        if( !(index < 0) ){
          nux = plane[numvp]->get_nux(*clayer.get_planeptr(ka));
          nuy = plane[numvp]->get_nuy(*clayer.get_planeptr(ka));
          shiftx = (clayer.plane[ka]->get_planex() - plane[numvp]->get_planex() -nux
+1)/2; // init x position
          shifty = (clayer.plane[ka]->get_planey() - plane[numvp]->get_planey() -nuy
+1)/2; // init y position

          for(int m=0; m<nux; m++){
            for(int n=0; n<nuy; n++){
              dtemp1 += plane[numvp]->get_weight(index, m*nux+n)*pow(clayer.plane[ka]
->get_celloutput(i+m+shiftx, j+n+shifty),2);
            }
          }

        } //end if
      } // ka
      dtemp1 = sqrt(dtemp1);
      plane[numvp]->cell[i][j].setforce_celloutput(dtemp1);
      dtemp1 = 0.0;
    } // j
  } // i

}


//for learning
void
Layer::response_stdSlayer(int x, int y, const Layer& clayer, Layer& vlayer, int numvp
, double r, double sigma)
{
  int index;
  int ctosnux, ctosnuy, vtosnux, vtosnuy;
  int shiftx, shifty;
  double dtemp1 =0.0;
  double dtemp2 =0.0;

  int  i = x;
  int  j = y;
```

```
  vlayer.output_stdVlayer(numvp, clayer);

  for(int k=0; k<numplane_; k++){
    for(int ka=0; ka<clayer.get_numplane(); ka++){

      index = plane[k]->find_connect(*clayer.get_planeptr(ka));
      if( !(index < 0) ){

        ctosnux = plane[k]->get_nux(*clayer.get_planeptr(ka));
        ctosnuy = plane[k]->get_nuy(*clayer.get_planeptr(ka));

        shiftx = (clayer.plane[ka]->get_planex() - plane[k]->get_planex() - ctosnux +
1)/2; // init x position
        shifty = (clayer.plane[ka]->get_planey() - plane[k]->get_planey() - ctosnuy +
1)/2; // init y position

        for(int m=0; m<ctosnux; m++){
          for(int n=0; n<ctosnuy; n++){
            dtemp1 += plane[k]->get_weight(index, m*ctosnux+n)*clayer.plane[ka]->get_
celloutput(i+m+shiftx, j+n+shifty);
          } // m
        } // n

      } //end if
    } // ka
    dtemp1 += sigma;
    // end of calculating numerator

    // calculate denominator
    index = plane[k]->find_connect(*vlayer.get_planeptr(numvp));

    vtosnux = plane[k]->get_nux(*vlayer.get_planeptr(numvp));
    vtosnuy = plane[k]->get_nuy(*vlayer.get_planeptr(numvp));

    for(int m=0; m<vtosnux; m++){
      for(int n=0; n<vtosnuy; n++){
        dtemp2 = sigma + r/(1.0 + r)*plane[k]->get_weight(index, 0)*vlayer.plane[numv
p]->cell[i][j].get_celloutput();
      }
    }
    // end of calculating denominator

    dtemp1 = r*phif(dtemp1/dtemp2 - 1.0);
    plane[k]->cell[i][j].setforce_celloutput(dtemp1);
    dtemp1 = 0.0;
  } // k

}

Plane*
Layer::mk_ssp(const Layer& clayer, int nux, int nuy) const
{
  Plane* tempplane;
  int shiftx, shifty;
  double temp =0.0;

  tempplane = new Plane(defaultpx_, defaultpy_, 0.0);

  for(int i=0; i<tempplane->get_planex(); i++){
    for(int j=0; j<tempplane->get_planey(); j++){

      for(int ka=0; ka<clayer.get_numplane(); ka++){

        shiftx = (clayer.plane[ka]->get_planex() - tempplane->get_planex() - nux +1)/
2;
```



```
        shifty = (clayer.plane[ka]->get_planey() - tempplane->get_planex() - nuy +1)/
2;

        for(int m=0; m<nux; m++){
          for(int n=0; n<nuy; n++){
            temp += clayer.plane[ka]->get_celloutput(i+m+shiftx,j+n+shifty);
          }
        }

      }
      tempplane->cell[i][j].setforce_celloutput(temp);
      temp = 0.0;
    }
  }

  return tempplane;
}

int
Layer::find_max_firing_plane(int x, int y) const
{
  int index =-1;
  double dtemp = 0.0;

  for(int k=0; k<get_numplane(); k++){
    if( plane[k]->get_celloutput(x,y) > dtemp ){
      index = k;
      dtemp = plane[k]->get_celloutput(x,y);
    }
  }

  return index;
}

void
Layer::learning_excite(int nump, const Layer& prvclayer, int cx_index, int cy_index,
double q, double(*weightc)(double x, double y))
{
  int prvctosnux, prvctosnuy;
  int shiftx, shifty;
  double dtemp;

  //change excitatory weights
  for(int ka=0; ka<prvclayer.get_numplane(); ka++){

    //initialize connect positons
    prvctosnux = plane[nump]->get_nux(*prvclayer.get_planeptr(ka));
    prvctosnuy = plane[nump]->get_nuy(*prvclayer.get_planeptr(ka));

    shiftx = (prvclayer.plane[ka]->get_planex() - plane[nump]->get_planex() - prvctos
nux +1)/2;
    shifty = (prvclayer.plane[ka]->get_planey() - plane[nump]->get_planey() - prvctos
nuy +1)/2;

    for(int m=0; m<prvctosnux; m++){
      for(int n=0; n<prvctosnuy; n++){
        dtemp = q*(*weightc)(m-prvctosnux/2, n-prvctosnuy/2)*(prvclayer.plane[ka]->ge
t_celloutput(m+cx_index+shiftx, n+cy_index+shifty));
        plane[nump]->delta_weight(*prvclayer.get_planeptr(ka), dtemp, m*prvctosnux+n)
;
      }
    }
  }

}
```

```
void
Layer::learning_inhibit(int nump, const Layer& vlayer, int numvp, int vx_index, int v
y_index, double q)
{
  double dtemp;

  int vtosnux = plane[nump]->get_nux(*vlayer.get_planeptr(numvp));
  int vtosnuy = plane[nump]->get_nuy(*vlayer.get_planeptr(numvp));

  for(int m=0; m<vtosnux; m++){
    for(int n=0; n<vtosnuy; n++){
      dtemp = q*vlayer.plane[numvp]->get_celloutput(m+vx_index-(vtosnux-1)/2, n+vy_in
dex-(vtosnuy-1)/2);
      plane[nump]->delta_weight(*vlayer.get_planeptr(numvp), dtemp, 0);
    }
  }

}

void
Layer::print_pgm(char* name) const
{
  strstream ss;
  char filename[100];

  for(int i=0; i<numplane_; i++){
    ss << name << "_planeNo" << i << ".pgm" << ends;
    ss >> filename;
    plane[i]->print_pgm(filename);
    ss.clear();
  }
}

void
Layer::print_data(char* name) const
{
  strstream ss;
  char filename[100];

  for(int i=0; i<numplane_; i++){
    ss << name << "_plane" << i << ".txt" << ends;
    ss >> filename;
    plane[i]->print_data(filename);
    ss.clear();
  }
}

void
Layer::print_pgm(char* name, int index1, int index2, int index3) const
{
  strstream ss;
  char filename[255];

  for(int i=0; i<numplane_; i++){
    ss << name << "_" << index1 << "_" << index2 << "_" << index3
       << "_planeNo" << i << ".pgm" << ends;
    ss >> filename;
    plane[i]->print_pgm(filename);
    ss.clear();
  }
}

void
Layer::print_data(char* name, int index1, int index2, int index3) const
{
  strstream ss;
```



```
  char filename[100];

  for(int i=0; i<numplane_; i++){
    ss << name << "_" << index1 << "_" << index2 << "_" << index3
       << "_planeNo" << i << ".txt" << ends;
    ss >> filename;
    plane[i]->print_data(filename);
    ss.clear();
  }
}

void
Layer::print_pgm_data(char* name, int index1, int index2, int index3) const
{
  print_pgm(name, index1, index2, index3);
  print_data(name, index1, index2, index3);
}

void
Layer::printall_pgm(char* name, int numx, int index1, int index2) const
{
  ofstream fout;
  strstream ss;
  char filename[255];
  int maxpx =0, maxpy =0;
  double max =0.0;

  ss << name << "_" << index1 << "_" << index2 <<".pgm" <<ends;
  ss >> filename;
  ss.clear();

  fout.open(filename);
  if(fout.fail()){
    cerr << "cannot open output file" << endl;
    exit(0);
  }

  for(int k=0; k<get_numplane(); k++){
    for(int i=0; i<plane[k]->get_planex(); i++){
      for(int j=0; j<plane[k]->get_planey(); j++){
        if( plane[k]->get_celloutput(i,j) > max )
          max = plane[k]->get_celloutput(i,j);
      }
    }
    if( plane[k]->get_planex() > maxpx ) maxpx = plane[k]->get_planex();
    if( plane[k]->get_planey() > maxpy ) maxpy = plane[k]->get_planey();
  }

  fout << "P2 " << (maxpx+2)*numx << " ";

  if( numplane_%numx == 0 ) fout << (numplane_/numx)*(maxpy+2) << " 255\n";
  else fout << (numplane_/numx+1)*(maxpy+2) << " 255\n";

  fout << "# true firing max is " << max <<"\n";
  for(int k=0; k<numplane_; k+=numx){
    for(int i=0; i<plane[k]->get_planex(); i++){
      for(int nx=0; nx<numx; nx++){
        if( !(k+nx < numplane_) ){
          for(int j=0; j<maxpx; j++){
            fout << setw(3) << 0 <<" ";
          }fout << "  0   0 ";
        }
        else{
          for(int j=0; j<plane[k+nx]->get_planey(); j++){
            if( max == 0.0 ) fout << setw(3) << 255 << " ";
            else
```

```
                fout << setw(3) << (int)(255.0 - plane[k+nx]->get_celloutput(i,j)/max*2
55.0) << " ";
            }fout << "  0    0 ";
          }

        }fout << "\n";
      }
      for(int i=0; i<2; i++){
        for(int j=0; j<(maxpx+2)*numx; j++){
          fout << setw(3) << 0 <<" ";
        }fout << "\n";
      }

    }
    fout.close();
}

void
Layer::printall_data(char* name, int index1, int index2) const
{
    ofstream fout;
    strstream ss;
    char filename[255];

    ss << name << "_" << index1 << "_" << index2 << ".txt" <<ends;
    ss >> filename;
    ss.clear();

    fout.open(filename);
    if(fout.fail()){
      cerr << "cannot open output file" << endl;
      exit(0);
    }

    for(int k=0; k<numplane_; k++){
      fout << "Plane No. " << k << "\n";
      for(int i=0; i<plane[k]->get_planex(); i++){
        for(int j=0; j<plane[k]->get_planey(); j++){
          fout << i << " " << j << " "
               << setw(10) << plane[k]->get_celloutput(i,j) << "\n";
        }fout << "\n";
      }fout<< "\n";
    }
    fout.close();
}

void
Layer::printall_pgm_data(char* name, int numx, int index1, int index2) const
{
    printall_pgm(name, numx, index1, index2);
    printall_data(name, index1, index2);
}
```

```
// -*-c++-*-
// Class extended vector

#ifndef ___N_VECTOR_H___
#define ___N_VECTOR_H___

#include <vector>

template <class Parts>
class n_vector
{
private:
  int number_;
  vector<Parts> element_;

public:
  n_vector(void);
  virtual ~n_vector(void);
  n_vector(const n_vector& x);
  n_vector& operator=(const n_vector& x);

  //methods
  virtual int get_number(void) const;
  virtual Parts get_element(int n) const;
  virtual void add_element(Parts input);
  virtual void rm_element(int n);
  virtual void set_element(int n, Parts input);
  virtual void delta_element(int n, Parts delta);
};

template <class Parts>
inline n_vector<Parts>::n_vector(void)
  :number_(0)
{
  element_.clear();
}

template <class Parts>
inline n_vector<Parts>::~n_vector(void)
{
}

template <class Parts>
inline n_vector<Parts>::n_vector(const n_vector& x)
  :number_(x.number_)
{
  element_ = x.element_;
}

template <class Parts>
inline n_vector<Parts>& n_vector<Parts>::operator=(const n_vector& x)
{
  if( &x != this ){
    number_ = x.number_;
    element_ = x.element_;
  }
  return *this;
}

template <class Parts>
inline int n_vector<Parts>::get_number(void) const
{
  return number_;
}

template <class Parts>
```



```
inline Parts n_vector<Parts>::get_element(int n) const
{
  return element_[n];
}

template <class Parts>
inline void n_vector<Parts>::add_element(Parts input)
{
  number_++;
  element_.push_back(input);
}

template <class Parts>
inline void n_vector<Parts>::rm_element(int n)
{
  if( !(n < 0) && n < number_ ){
    number_--;
    element_.erase(element_.begin()+n);
  }
  else{
    cerr << "warning : out of range, index of element" << "\n";
  }
}

template <class Parts>
inline void n_vector<Parts>::set_element(int n, Parts input)
{
  if( !(n < 0) && n < number_ ) element_[n] = input;
  else{
    cerr << number_;
    cerr << "warning : out of range, index of element " << n << "\n";
  }
}

template <class Parts>
inline void n_vector<Parts>::delta_element(int n, Parts delta)
{
  if( !(n < 0) && n < number_ ) element_[n] += delta;
  else{
    cerr << "warning : out of range, index of element " << n << "\n";
  }
}

#endif // ___N_VECTOR_H___
```

参考文献

[1] 福島邦彦:“神経回路と情報処理”, 朝倉書店,1989.

[2] K.Fukushima,S.Miyake:“Neocognitron : A Hierarchical Neural Network Capable of Visual Pattern Recognition”,Neural Networks,1(2),pp.119-130,1988.

[3] 佐藤俊治, 黒岩丈介, 阿曽弘具, 三宅章吾:“回転対応型ネオコグニトロン”, 電子情報学会論文誌,J81-D-II,6,pp.1365-1374,1998.

[4] M.Kikuchi,K.Fukushima:“Neural Network Model of the Visual System:Binding Form and Motion”,Neural Networks,9(8),pp.1417-1427,1996.

[5] K.Fukushima:“Neural network model for selective attention in visual pattern recognition and associative recall”,Applied Optics,26,23,pp4985-4992,1987.

[6] 甘利俊一:“神経回路網の数理”, 産業図書株式会社,pp.7-13,1978.

[7] 伊藤正男:“脳の不思議”, 岩波書店,1998.

[8] Neil R. Carlson:“Physiology of Behavior”,Allyn and Bacon,1998.

[9] 和家伸明, 福島邦彦:“ネオコグニトロンの新しい学習法”, 電子情報学会論文誌,J75-D-II,11,pp.1892-1899,1992.

[10] $\varepsilon\pi\iota\sigma\tau\eta\mu\eta$:“Standard Template Library プログラミング”, 秀和システムズ,pp.49,1996.

[11] Scott Meyers, 古川邦夫:“Effective C++”, アスキー出版局,1998.

[12] 岡崎哲郎, 庄野逸, 福島邦彦:“ネオコグニトロン型神経回路モデルのシミュレーションを対象としたライブラリの設計と実装”, 電子情報通信学会技術研究報告,NC96-1117,1997.

[13] 佐藤俊治:“階層的パターン統合処理に基づく視覚情報処理モデルに関する研究”, 東北大学審査博士学位論文,2000.

発表予定学会

宮野靖弘，佐藤俊治，阿曽弘具，三宅章吾，
“ネオコグニトロン型神経回路モデルの開発支援を目的としたライブラリの設計と実装”，
電子情報通信学会ニューロコンピューティング研究会，2000 年 3 月，於玉川大学.

謝辞

本研究を行うにあたって，数多くの御指導とともにこの研究の機会を与えて下さった，東北大学大学院工学研究科 阿曽 弘具 教授に心から感謝いたします．ニューラルネットワークの研究分野全般において，貴重な御意見や議論を交して頂きました，東北大学大学院工学研究科応用物理学科 三宅 章吾 講師に深く感謝いたします．研究方針について貴重な時間をさいて御指導して下さった東北大学大学院工学研究科 大町 真一郎 助教授に深く感謝いたします．

本論文をまとめるにあたり，貴重な御意見を頂きました東北大学大学院工学研究科 阿部 健一 教授，東北大学電気通信研究所 白鳥 則郎 教授に心から感謝いたします．

日々の研究につきましては，いつでも心良く議論を交して下さった，東北大学大学院工学研究科 佐藤 俊治 氏，菅谷 至寛 氏，また，東北大学電気通信研究所 伊藤 真 氏に心から感謝いたします．東北大学大学院工学研究科 阿曽研究室 高田 直幸 氏，須藤 貴志 氏，下村 正夫 氏をはじめとする，阿曽研究室の皆様には，貴重な御意見，御討論をいただきましたことに深く感謝いたします．

修士論文審査

ＯＨＰ資料

ニューラルネットワークの開発支援システムに関する研究

東北大学 大学院 工学研究科 電気・通信工学専攻
阿曽研究室 博士前期の課程2年 宮野靖弘

第1章 序論

- 脳の高度な情報処理を解明する研究
  - 生理学的、心理学的研究
  - 工学的研究
    - 生理学的事実をもとにニューラルネットモデルを作成し
      モデルの動作から脳を解明

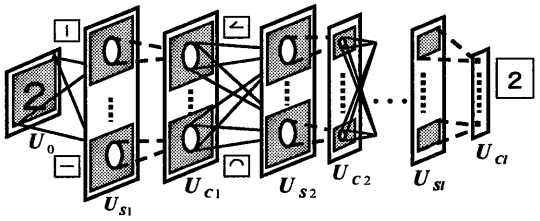
脳における視覚情報処理 ・生理学知見が比較的明らか

視覚系神経回路モデルによるパターン認識の研究
数多くのモデルが提案されている

視覚系神経回路モデル

ネオコグニトロン(neocognitron) Fukushima 1980

- 神経生理学で得られた事実を採用
- パターン認識などに用いられている



ネオコグニトロンを基礎とした多種、多様、高機能なモデル
構造が複雑化

⇩

プログラミングに要する時間が膨大
追加実験が困難

3

過去の研究

ネオコグニトロン型神経回路モデルのシミュレーションを対象とした
ライブラリの設計と実装(岡崎ら 1997)

- ネオコグニトロン型モデルに限定している
- 拡張性が乏しい
- 新型モデルへの考慮がなされていない

4

目的

ニューラルネットワークの開発支援システムの作成
ネオコグニトロン型モデルを対象とするライブラリの設計

- 共通部分のライブラリ化
- モデル作成におけるプログラミング時間の短縮
  - 新モデルの作成を容易にする
  - 追実験を容易にする
- 他のニューラルネットモデルへの対応を考慮

5

本論文の構成

第1章 序論

第2章 ネオコグニトロン

構造および学習の仕組み

第3章 ライブラリ設計

設計方針、ライブラリの構造

第4章 ライブラリの評価

モデルの構築、プログラミング量の評価

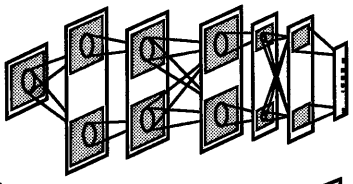
第5章 結論

6

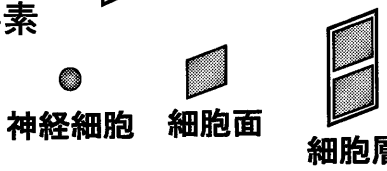
第2章 ネオコグニトロン

パターン認識を行う視覚神経系回路モデル

•多層構造

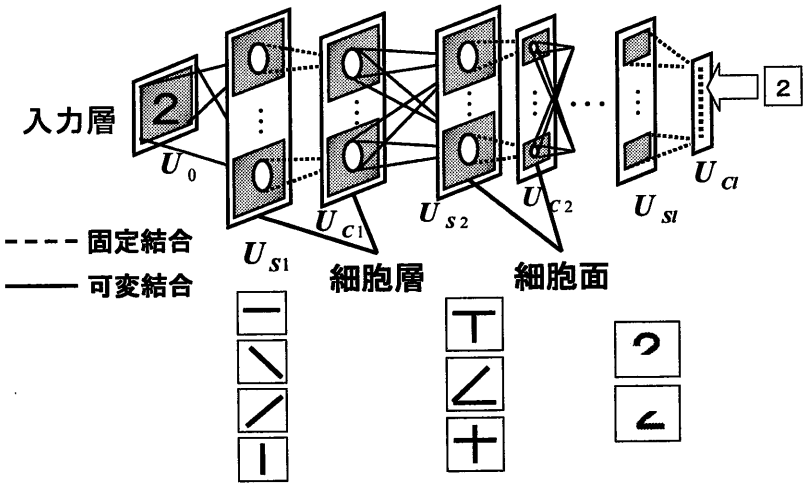


•構成要素



7

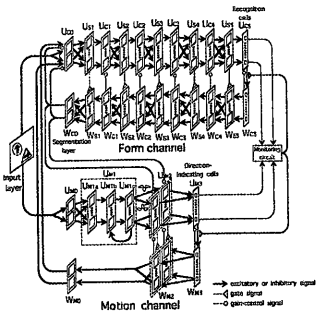
ネオコグニトロン全体図

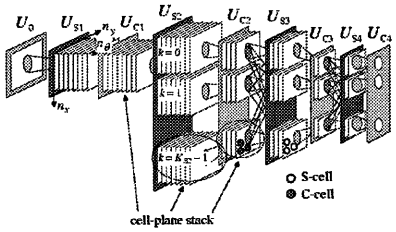


8

ネオコグニトロンに基礎をおくモデル

選択的注意視と連想想起 (Fukushima 1987)
バインディング問題の解決 (Kikuchiら 1996)
回転したパターンの認識 (Satohら 1998)





9

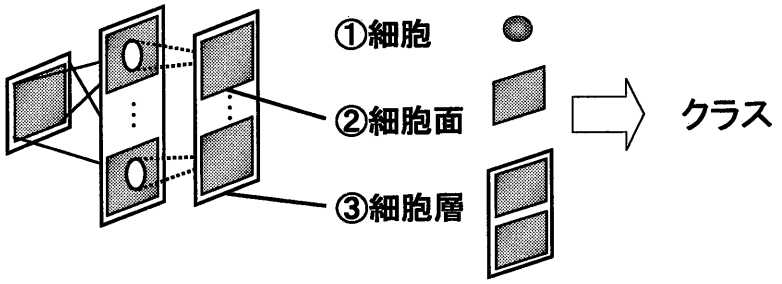
第3章 ライブラリ設計

ライブラリの設計方針

- 各構成要素を**階層的なクラス**としてライブラリに実装
  - ネオコグニトロン型モデルの構成要素間の階層関係を考慮
  - 新しいネオコグニトロン型モデルに柔軟に対応
  - 学習時におけるモデルの動的な構造変化に対応
- プログラム言語にC++を採用
  - 階層構造を持つデータの記述に有効
  - 再利用性の高いプログラミングが可能
- 計算機資源の有効利用

10

ネオコグニトロンの構成要素

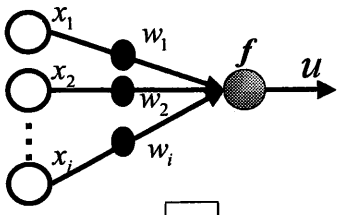


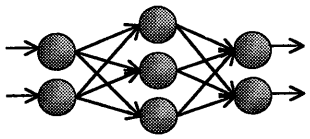
- クラス(class)
  - C++ のデータ構造
  - オブジェクトの型を定義

11

①細胞

多入力1出力の情報処理素子





x_i :入力信号
w_i :結合重み
f :発火関数
$u = f(w_1x_1 + \cdots + w_nx_n)$
:出力値

多数の細胞の集合・結合
- 細胞面
- 細胞層
- パーセプトロン

12

細胞のクラス

細胞Class Cell

```
class Cell
{
private:
 double celloutput_;
 ConnectWeight<Cell> connectw_;

public:
 ...
 virtual void mk_connect(const Cell& newcell, double initw =0.0);
 virtual void delta_weight(const Cell& cell, double deltaw, int n =0);
 ...
 virtual double output_sigmoid(double th, double coef);
```

細胞出力値

結合情報

①

②

③

代表的なメンバ関数

①細胞間の結合を形成するメンバ関数

使用例：mk_connect(細胞, 重み初期値);

②重みを更新するメンバ関数

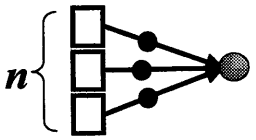
使用例：delta_weight(結合もとの細胞, 重みの増減);

③代表的な出力関数

結合情報部のクラス

結合情報Class① ConnectWeight

```
template <class Parts> class ConnectWeight
{
private:
 int numconnect_;
 vector< OneConnectWeight<Parts>* >* connect_;
...
```



結合数 n

OneConnectWeight格納用可変長配列

結合情報Class② OneConnectWeight

```
template <class Parts> class OneConnectWeight
{
private:
 int nux_, nuy_;
 const Parts* connect_;
 vector<double> weight_;
                ...
```

結合重み

一つの結合もとの位置

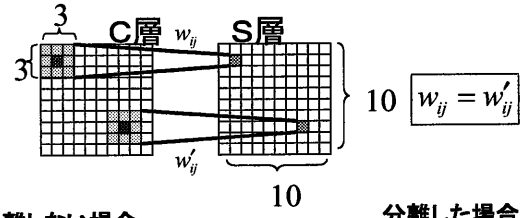
結合範囲

結合もと(ソース中ではParts)へのポインタ

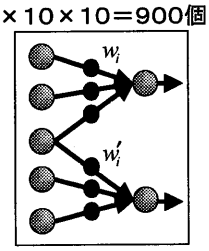
結合重み

細胞本体と結合情報部の分離による利点①

細胞面内の全ての細胞は同じ結合重み分布を持つ

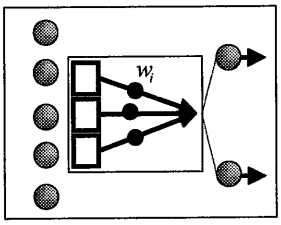


分離しない場合
3×3×10×10=900個の結合



効率改善

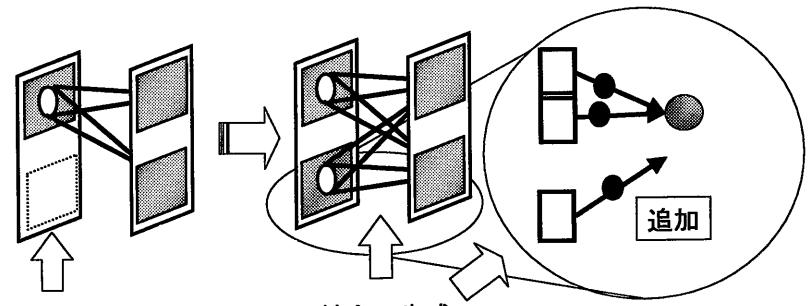
分離した場合
3×3=9個の結合



15

細胞本体と結合情報部の分離による利点②

構造の動的変化に柔軟に対応



16

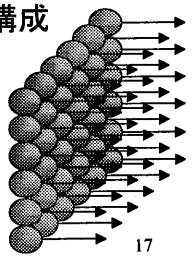
②細胞面の設計

同種の細胞を2次元平面上に並列

従来のライブラリ
- ネオコグニトロンにおける最小構成要素
- 細胞層のデータの一部
- 細胞の機能の簡略化

本ライブラリ
- 最小構成要素である細胞から細胞面を構成
- 細胞間の結合が任意に設定可

⇨ 細胞面間においても
自由度の高い結合が可能



17

細胞面のクラス

細胞面Class Plane

```
class Plane
{
private:
  int plane_x_, plane_y_;          ← 細胞面のサイズ
  ConnectWeight<Plane> connectw_;  ← 結合情報

public:
  Cell** cell;                     ← 細胞（2次元配列）
...
 virtual void mk_connect(const Plane& newplane, double initw, int nx, int ny);
...
```

代表的なメンバ関数
細胞面間の結合を形成するメンバ関数
使用例: mk_connect(細胞面, 重み初期値, 結合範囲x, y);

18

③細胞層の設計

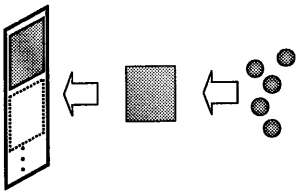
細胞層Class Layer

```
class Layer
{
private:
  const int defaultpx_, defaultpy_;
  const double defaultinitcell_;
  int numplane_;
  ConnectWeight<Layer> connectw_;

public:
  vector<Plane*> plane;
  ・・・
  virtual int add_plane(void);
  ・・・
```

細胞層

使用例：Layer(細胞面数, 細胞面サイズx,y);

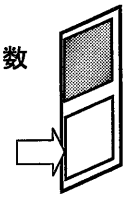


細胞面数

Plane格納用可変長配列

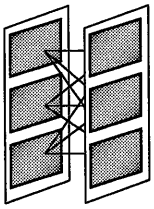
細胞面増加関数
add_plane( );

細胞面増加



細胞層間の結合

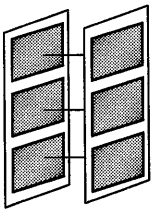
代表的な結合の関数化



多対多

C－S結合
V－S結合 } mk_stdSconnect(C細胞層, V細胞層, 結合範囲x,y等・・・);

C－V結合 mk_stdVconnect(C細胞層, 結合範囲x,y);



1対1

S－C結合 mk_stdCconnect(S細胞層, 結合範囲x,y);

学習則のライブラリ化

学習則はモデルを特徴づける独自のものである
- それぞれの研究者が作成する

<u>一般的な学習則はライブラリに必要である</u>

シードセル生成面を用いた学習法（Wakeら 1992）
- 未学習の特徴を見つけると細胞面を自動的に増やし その特徴を学習する

21

第3章 まとめ

ネオコグニトロンの階層構造を実現したライブラリを提案
- ネオコグニトロン型モデルの構造と一致
- 直感的なライブラリ構造
- モデルの動的な構造変化に対応
- 計算機資源の効率的な使用

ライブラリのソースコード量

細胞に関して	255行
細胞面に関して	401行
細胞層に関して	921行
結合情報に関して	510行
学習に関して	117行
その他	60行
合計	2206行

22

第4章 ライブラリの評価

評価対象
　ライブラリを用いて記述したネオコグニトロン

評価基準
　実際のプログラミング量
　学習、認識動作の検証

その他
　ライブラリの応用例

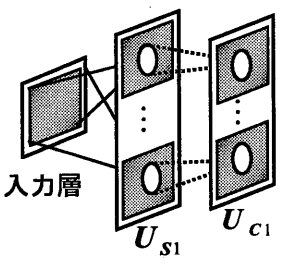
ネオコグニトロン型モデル：構造設計

全体の構造

```
//ネオコグニトロンの構造
 //入力層
 Layer C0(1, SIZEC0,SIZEC0);
 //第1段
 Layer V1(1, 19,19);
 Layer S1(12, 19,19);
 Layer C1(12, 21,21);
 //第2段
 Layer V2(0, 21,21);
 Layer S2(0, 21,21);
 Layer C2(0, 13,13);
 //第3段
 Layer V3(0, 13,13);
 Layer S3(0, 13,13);
 Layer C3(0, 7,7);
 //第4段
 Layer V4(0, 3,3);
 Layer S4(0, 3,3);
 Layer C4(0, 1,1);
```

U_{S1}とU_{C1}の構築

```
//第1S層、C層----------
 S1.mk_stdSconnect(C0, 0.1, V1, 0, 100000, 3,3);
          /第1層 C0→S1層間結合確立
 C1.mk_stdCconnect(S1, 3,3);
          //第1層 S1→C1層間結合確立
 set_initw1(C0, S1); //C0→S1層間の重みの初期化
```



ネオコグニトロン型モデル：学習

```
//入力層にデータ入力
C0.plane[0]->set_input(d_data);

//1段目----------
 S1.output_stdSlayer(C0, V1, 0, R1);
 C1.output_stdClayer(S1);

//2段目----------
if( stage == 0 ){
 learning1(C1, S2, 0.1, V2, 0.1, 3,3, C2, 3,3, 100000, R2);
}
else{
 S2.output_stdSlayer(C1, V2, 0, R2);
 C2.output_stdClayer(S2);
}

//3段目----------
if( stage == 1 ){
 learning1(C2, S3, 0.1, V3, 0.1, 5,5 ,C3, 3,3, 100000, R3);
}
else if( stage > 1 ){
 S3.output_stdSlayer(C2, V3, 0, R3);
 C3.output_stdClayer(S3);
}

//4段目----------
 learning1(C3, S4, 0.1, V4, 0.1, 3,3 ,C4, 3,3, 100000, R4);
```

入力層にパターンを提示

1段目 U_{S1} U_{C1} の出力計算

2段目 U_{S2} U_{c2} の学習

3段目 U_{S3} U_{c3} の学習

4段目 U_{S4} U_{c4} の学習

25

ネオコグニトロン型モデル：認識

```
//入力層にデータ入力
C0.plane[0]->set_input(d_data);

S1.output_stdSlayer(C0, V1, 0, R1);
C1.output_stdClayer(S1);

S2.output_stdSlayer(C1, V2, 0, R2);
C2.output_stdClayer(S2);

S3.output_stdSlayer(C2, V3, 0, R3);
C3.output_stdClayer(S3);

S4.output_stdSlayer(C3, V4, 0, R4);
C4.output_stdClayer(S4);

rec_count[C4.find_max_firing_plane(0,0)]++;
```

入力層にパターンを提示

1段目 U_{S1} U_{C1} の出力計算

2段目 U_{S2} U_{c2} の出力計算

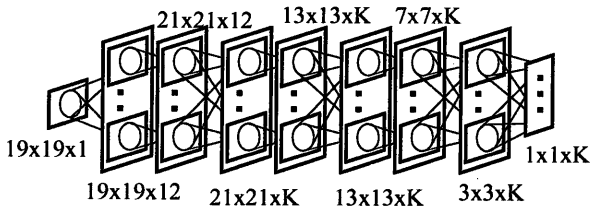
3段目 U_{S3} U_{c3} の出力計算

4段目 U_{S4} U_{c4} の出力計算

main関数
ネオコグニトロンの構造、動作 223行

26

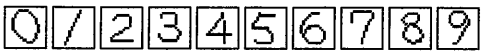
動作検証



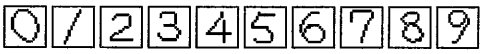
K：学習により決定する

⇒ 数字の学習・認識実験

学習

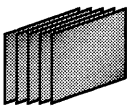


認識



27

拡張されたモデルへの対応

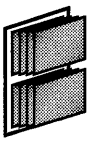


細胞面群

- 複数の細胞面により構成

継承により対応

```
class PlaneStack : public Layer
{
public:
  PlaneStack(int nump, int px, int py, double initcell =0.0);
};
```



細胞面群を伴った細胞層

- 細胞面群および細胞層により構成

構成要素の組み合わせで対応

```
class LayerPS
{
private:
  int numplanestack_;
  ConnectWeight<LayerPS> connectw_;

public:
  vector<PlaneStack*> planestack_;
...
```

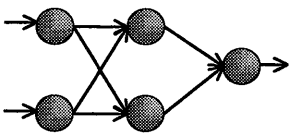
28

その他のニューラルネットモデルへの対応

例:パーセプトロンへの応用

```
Cell input_unit[2];
Cell hidden_unit[2];
Cell output_unit;

//結合形成
for(int i=0; i<2; i++){
  for(int j=0; j<2; j++){
    //入力層→隠れ層
    hidden_unit[i].mk_connect(input_unit[j],rand());
  }
  //隠れ層→出力層
  output_unit.mk_connect(hidden_unit[i], rand());
}
```



29

第5章 結論

まとめ

- ネオコグニトロンの階層構造を実現したライブラリを提案
  - ネオコグニトロン型モデルの階層構造と一致
  - 直感的なライブラリ構造
  - モデルの動的な構造変化に対応
  - 計算機資源の効率的な使用
  - モデル構築時のプログラム量の削減

今後の課題

- より多くのネオコグニトロン型モデルの動作検証
- 利便性の追求

30