TR-I-0119

ニューラルネットにおけるバックプロパゲーション学習の効率化方法

A new method to speed up the Back-Propagation algorithm

中村雅己、鹿野清宏

M. Nakamura, K. Shikano

1989.10

概要

ニューラルネットを用いたモデルの適用が活発になってきたが、適用問題の本格化、高度化にともない、用いるニューラルネットのネットワーク構造が複雑で大規模なものが必要となり、また、学習データも大量に必要になるため、学習時間の問題が切実なものとなった。この問題を解決するために、我々はバックプロパゲーション法を用いた多層ニューラルネットの学習において、学習の程度を決定するパラメータを動的に変化させることにより、学習の効率化を図る方法を考案した。

本報告では、まず、バックプロパゲーション学習法の概要について述べ、学習の効率化方法および比較実験結果を報告する。また、付録に学習効率化シミュレーションプログラム(DCP2)の取扱い説明、仕様を載せる。

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

目次

1 はじめに

ニューラルネット、特に多層ネットワークにバックプロパゲーション学習アルゴリズム[1)2)]を用いたモデルの応用が活発になってきている。音声等のパターン認識[3)]の問題を中心に、アナログ信号の写像の問題[5)]や記号処理の分野[6)7)]まで適用されつつある。このようにニューラルネットの適用が多方面でなされるにつれ、その学習アルゴリズムの問題が顕在化しはじめた。すなわち、適用問題の本格化、高度化にともない、用いるニューラルネットのネットワーク構造が複雑で大規模なものが必要となり、また、学習データも大量に必要になるため、学習時間の問題が切実なものとなった。現在、大規模なパラレルプロセッシングが可能なコネクションマシンは開発途上にあるため、現実的にはシングルプロセッサあるいは小規模のパラレルプロセッサにより学習を行っているのが大半であり、学習に非常に多くの時間がかかってしまう。この問題を解決することは、現状におけるニューラルネットの応用研究を進める上で、極めて大切なことである。また、学習に用いるパラメータも問題に応じて試行錯誤で決定しているのが現状であり、最適なパラメータを問題に応じて自動的に決定する方法を考えることは、将来の大規模なパラレルプロセッシングの時代においても非常に有益なことである。

そこで、本報告ではバックプロパゲーション法を用いた多層ニューラルネットの学習において、学習の程度を決定するパラメータを動的に変化させることにより、学習の効率化を図る方法について述べる。まず、バックプロパゲーション学習法の概要について述べ、学習の効率化方法および比較実験結果を報告する。また、付録に学習効率化シミュレーションプログラム(DCP2)の取扱い説明、仕様を載せるので参考にしていただきたい。

2 バックプロパゲーション学習法の概要

バックプロパゲーション学習アルゴリズムが対象とするネットワークは入力層、出力層およびHidden-Layerと呼ばれる隠れ層で構成される多層ネットワークである(図1)。バックプロパゲーション学習アルゴリズムの特徴は、教師付き学習において出力層から入力層に誤差を伝搬させることにあり、これにより各ユニットにおいて最急降下法を適用することが可能となった。もう一つの特徴は、各ユニットに非線形関数を導入したことであり、これにより入力から出力への写像に、より一般性を持たせることができた。ここではバックプロパゲーション学習のアルゴリズムについて数式を追って説明する。

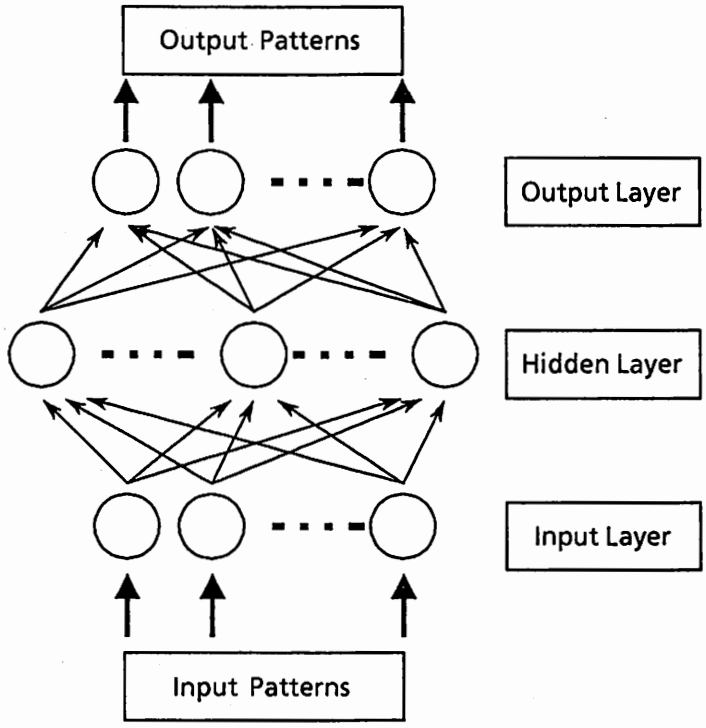
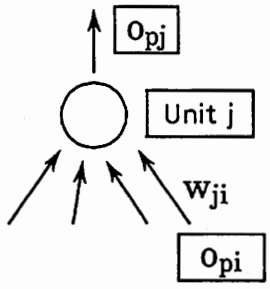


図1多層ニューラルネットワーク



$$o_{pj}=f(net_{pj})$$

$$net_{pj}=\sum_i w_{ji}\cdot o_{pi}$$

図2ニューラルネットワークのユニット

ネットワークのユニット間は入力層から出力層に向かって結合されており、入力層にある入力パターンpを入力したとき、(図2)に示すようにそれぞれのユニットjでは他のユニットiからの入力(すなわちユニットiの出力)と、ユニットi,j間の結合重み係数w_{ji}の積の総和$net_{pj}=\Sigma w_{ji}\cdot o_{pi}$をとり、さらに入出力関数f(x)を通して、出力信号$o_{pj}=f(net_{pj})$を出す。すなわち、ある入力信号のパターンをネットワークの入力層にいれたときに、上記のような計算を全てのユニットで行い(但し、入力層では入出力関数を通さないことが多い)、最終的に出力層から出た信号パターンが望ましいパターンになるように、ユニット間の結合重み係数を決定すればよい。

まず、この問題を最小化問題として定式化するために評価関数を決める。

ここでは教師信号とニューラルネットの出力信号の誤差の2乗和E_pを用いる。

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \qquad (1)$$

ここでt_{pj}は入力パターンpに対する出力ユニットjの教師信号であり、o_{pj}は出力ユニットjの出力信号である。この誤差関数E_pを全ての入力パターンに対して最小にする必要がある。よって問題は$E = \Sigma E_p$を最小にするような結合重み係数を決定するという、最小化問題となる。

この問題を解くために、バックプロパゲーション学習アルゴリズムでは最急降下法を用いている。すなわち、入力パターンp毎の結合重み係数w_{ji}の更新量$\Delta_p w_{ji}$を次式のようにエラー空間の勾配に比例した形で与える。

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}} \qquad (2)$$

まず、$\partial Ep/\partial wji$を求める。

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_{ji}} \qquad (3)$$

$net_{pj} = \Sigma w_{jk} \cdot o_{pk}$であるから右辺の右側は

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} \cdot o_{pk} = o_{pi} \qquad (4)$$

である。ここで添字kはユニットjに入力結合するすべてのユニットの番号である。次にユニットjに対して

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} \qquad (5)$$

とおくことにより、

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} \cdot o_{pi} \qquad (6)$$

よって、(2)式の学習規則は

$$\Delta_p w_{ji} = \eta \cdot \delta_{pj} \cdot o_{pi} \qquad (7)$$

となる。ここでηはステップサイズを決定する正の定数である。

次にδ_{pj}を求める必要がある。

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}} \qquad (8)$$

ここで、$o_{pj} = f(net_{pj})$であるから右辺の右側は

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'(net_{pj}) \qquad (9)$$

である。(8)式の右辺の左側はユニットjが出力ユニットか、そうでないかによって式は異なる。ユニットjが出力ユニットの場合、

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \qquad (1)$$

であるから、

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) \qquad (10)$$

となり、直接δ_{pj}が次式のように求まる。

$$\delta_{pj} = (t_{pj} - o_{pj}) \cdot f'(net_{pj}) \text{ (ユニットj;出力ユニット)} \qquad (11)$$

一方、ユニットjが出力ユニットでない場合、E_pがo_{pj}の直接の関数とならない。従って、次式のように変形してδ_{pj}の再帰関数として求めるという工夫を行う。

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial net_{pk}}{\partial o_{pj}}$$

$$= \sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial}{\partial o_{pj}} \left(\sum_i w_{ki} \cdot o_{pi}\right)$$

$$= \sum_k \frac{\partial E_p}{\partial net_{pk}} w_{kj} = -\sum_k \delta_{pk} w_{kj} \qquad (12)$$

$$\delta_{pj} = f'(net_{pj}) \cdot \sum_k \delta_{pk} w_{kj} \text{ (ユニットj;隠れユニット)} \qquad (13)$$

このように、$\Delta_p w_{ji}$を計算するのに必要な誤差情報δ_{pj}を出力層から入力層へ逆に伝搬して行くのでバックプロパゲーションという。

バックプロパゲーション学習アルゴリズムではユニットの入出力関数f(x)として、式(11)、(13)から明らかなように微分可能な関数が必要とされる。文献1)では次のような非線形単調増加のシグモイド関数を用いるのが良いとしている。

$$f(x) = \frac{1}{1 + e^{-x}} \qquad (14)$$

すなわち、ユニットjの出力は次式のようになる。

$$f(net_{pj}) = o_{pj} = \frac{1}{1 + e^{-net_{pj}}} \qquad (15)$$

ここで入力の総和net_{pj}は$net_{pj}=\Sigma w_{ji}\cdot o_{pi}+\theta_j$としてバイアス成分$\theta_j$を加える。実際のネットワークでは入力ユニット以外のすべてのユニットと結合する、出力が常に1のバイアスユニットを考え、θ_jをその結合重み係数とみなして学習する。

$f(net_{pj})$の導関数を求めると

$$f'(net_{pj})=\frac{\partial o_{pj}}{\partial net_{pj}}=o_{pj}\cdot(1-o_{pj}) \tag{16}$$

よって、結合重み係数w_{ji}の更新量$\Delta_p w_{ji}$は次の式で得られる。

$$\Delta_p w_{ji}=\eta\cdot\delta_{pj}\cdot o_{pi} \tag{17}$$

ただし、ユニットjが出力ユニットの場合は

$$\delta_{pj}=o_{pj}\cdot(1-o_{pj})\cdot(t_{pj}-o_{pj}) \tag{18}$$

であり、ユニットjが隠れユニットの場合は

$$\delta_{pj}=o_{pj}\cdot(1-o_{pj})\cdot\sum_k\delta_{pk}w_{kj} \tag{19}$$

である。

結合重み係数w_{ji}の更新は、入力パターンが複数個あるのが一般的であるので、1つの入力パターン提示毎に実行するか、次式のように全入力パターン提示後に

$$\Delta w_{ji}=\eta\cdot\sum_p(\delta_{pj}\cdot o_{pi}) \tag{20}$$

として実行するか、2つの方法がある。また、結合重み係数w_{ji}の初期値をすべて同じ値にすると、隠れユニットの出力値がすべて同じになり、学習が進まないので、乱数等によりランダムな小さな初期値に設定する必要がある。

以上がバックプロパゲーション法の概要である。処理フローを図3に示す。

この方法の基本原理は最急降下法であるため、最短距離で最小値に到達するためには、更新幅を無限小にする必要があるが、実際問題として、計算繰り返し回数が増加するため、収束速度は遅くなる。そこで、なるべく大きな更新幅(Δw_{ji})を得るために(7)式のηの値を大きくとりたいのであるが、更新方向が振動し易くなる。文献1)では、前回の更新幅をモーメンタム量として次式のように加算することにより振動を抑制することを提案している。

$$\Delta w_{ji(n+1)}=\eta\cdot\delta_{pj}\cdot o_{pi}+\alpha\cdot\Delta w_{ji(n)} \tag{21}$$

ここで、αはモーメンタム量を調整するパラメータである。このη,αは定数

であるが、これらの最適な値(収束が早くなる値)はエラー空間の形状、すなわちタスクの種類やサンプルデータの量によって異なり、さらに学習進行の程度によっても変化する。

3 学習効率化方法(DCP法)

従来、パラメータη,αの値は学習するタスクに応じて経験的に決めていた。このため、学習に時間がかかったり、不適当な局所的最小値に陥りやすいといった問題点があった。

そこでこの問題を解決するために、学習繰り返し計算毎、若しくは何回かの学習繰り返し計算に1回の割合で、次式によりエラーE_pが最小となるようにη,αをダイナミックに変更する方法を考案した。(Dynamic Control training Parameters : DCP 法)

$$E_p(w_{ji(k)} + \Delta w_{ji(k)(\eta(k),\alpha(k))}) \qquad (22)$$
$$= \underset{l,m}{\mathrm{Min}}\ E_p(w_{ji(k)} + \Delta w_{ji(k)(\eta l,\alpha m)})$$

実際にはη,α の値を有限個用意して、その中からE_pが最も小さくなるη,αを選択する。ただし、η については、例えば前回最適値の1/2、1、2倍の値から選択することで、繰り返し学習の結果、ηの値を増加あるいは減少させることができる。DCP 法の処理フローを図4に示す。

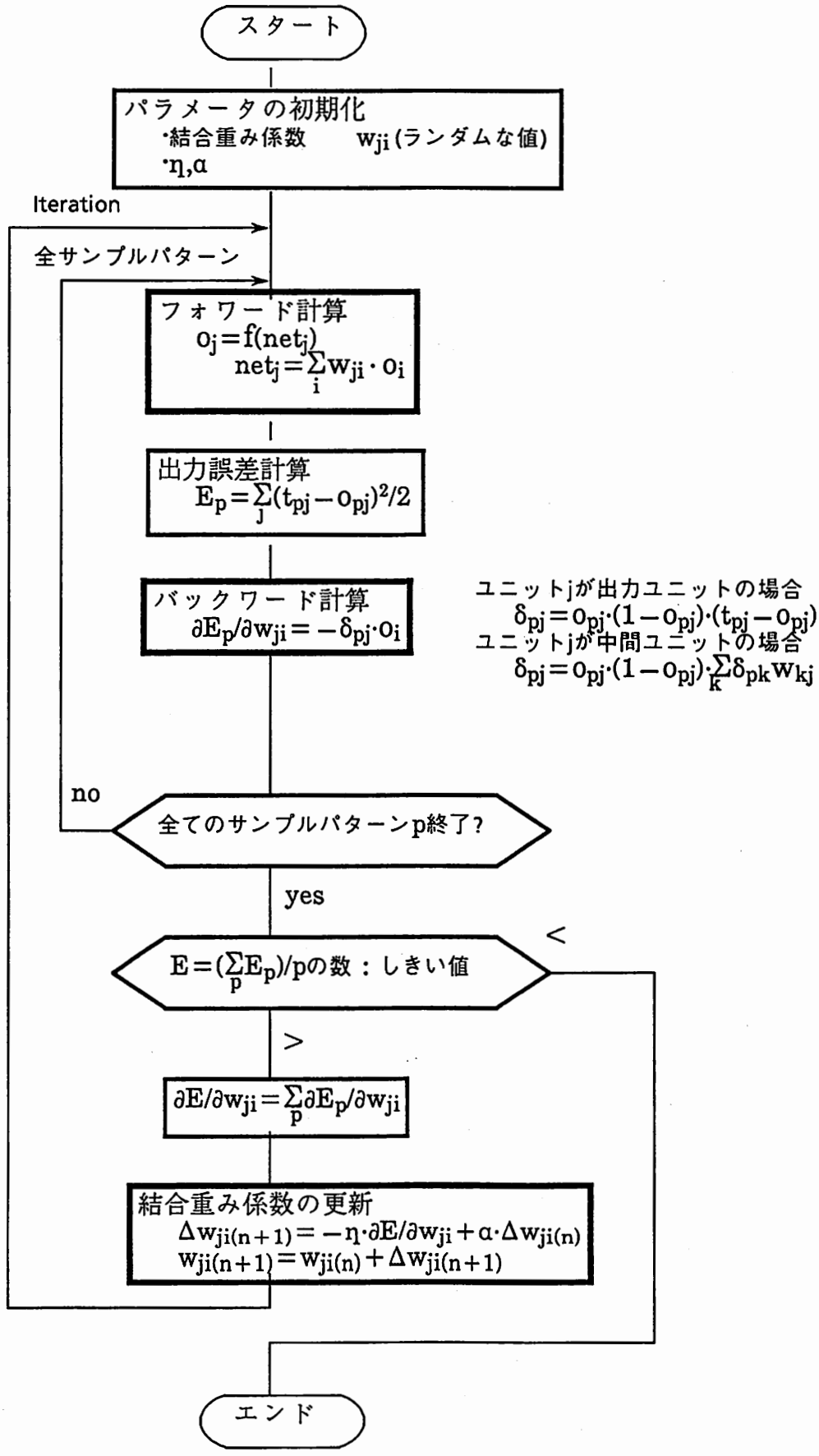


図3　一般的なバックプロパゲーション学習の手順

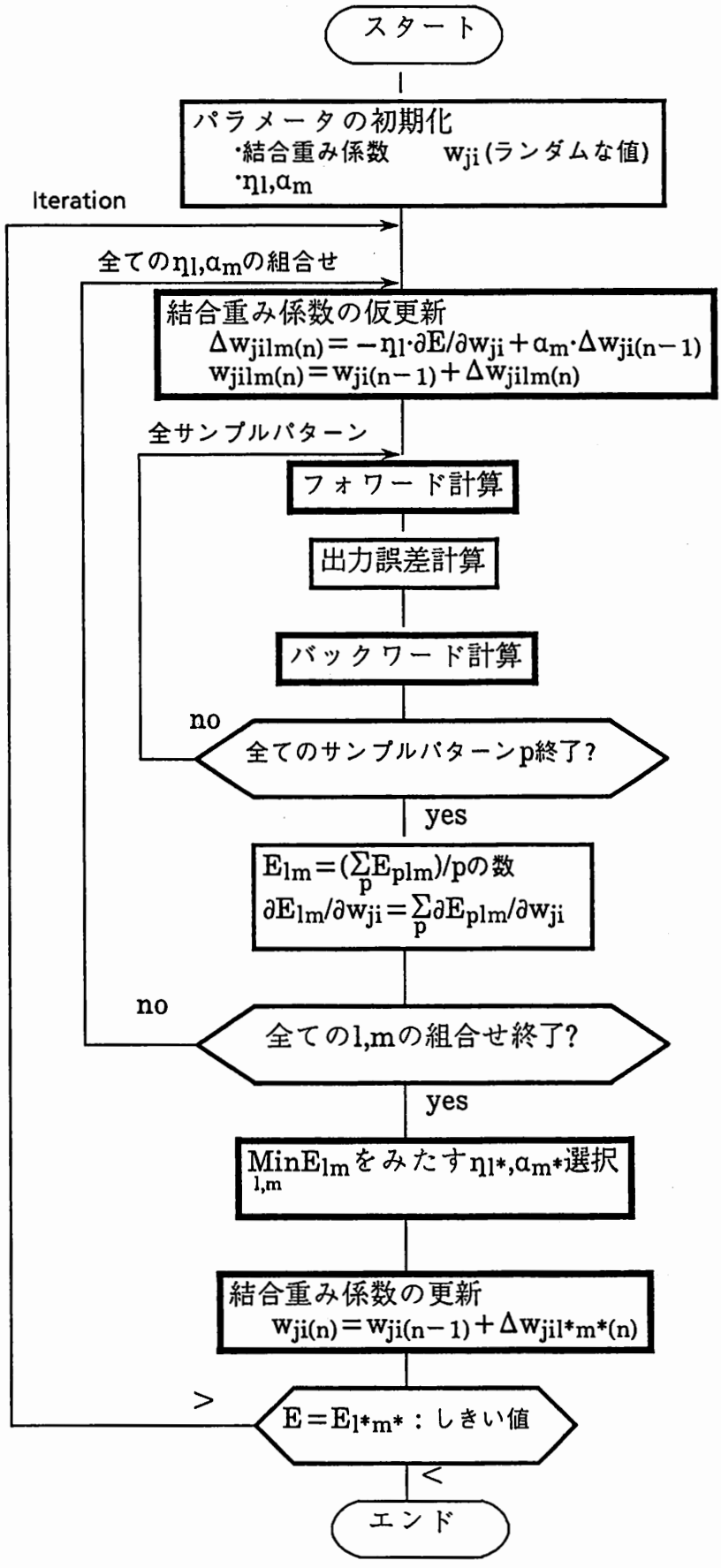


図4　DCP法によるバックプロパゲーション学習の手順

4 効果確認実験および考察

DCP法の効果を調べるために従来法との比較実験を行った。実験条件は次のとうりである。

(1) タスク　　単語列予測モデル(NETgram)[7]

(2) ネットワーク入力　　現在の単語の品詞番号
　　(品詞番号に相当するユニットのみ1で他はすべて0)

(3) ネットワーク出力教師信号　　次の単語の品詞番号

(4) ネットワーク構造(図－5)
　　入力層　　89個のユニット(品詞の数)＋1個(しきいユニット)
　　中間層　　16個のユニットが2層
　　出力層　　89個のユニット(品詞の数)

(5) サンプル数　23(1センテンス)

(6) パラメータ

ステップ幅η　次のいずれか
　　○(従来方法)　一定値(0.1 or 0.4)
　　○(DCP方法)　(1/2, 1, 2)×η(k－1)
　　(前回値の1/2,1,2倍の3種類の選択)

モーメンタムα　次のいずれか
　　○(従来方法)　一定値(0 or 0.9)
　　○(DCP方法)　(0.0, 0.9)(2種類の選択)

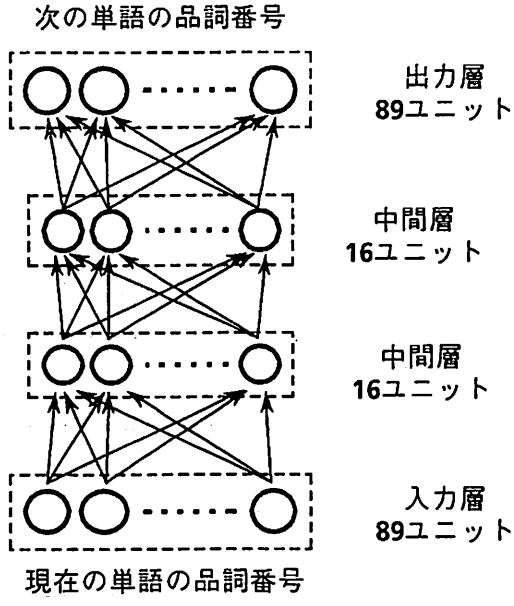


図5　単語列予測モデルの構造

実験結果を表ー1、図6にしめす。

従来のパラメーター定に比較して、η,αをダイナミックに変更する方法は約4.3倍以上収束が早くなっている。(CASE1,CASE2)

また、パラメーター定の場合、η=0.4のときはエラーの振動が生じ易く(CASE3,CASE4)、η=0.1であれば収束がおそくなる(CASE2,CASE5)。αについてはα=0.9の場合、ηの値も大きければ不安定な状態が持続し(CASE3)、α=0の場合、学習が進んでも収束速度が加速しない(CASE5)。

次に本方法(η,αをダイナミックに変更する方法)により、上記タスクに対しエラーの収束判定で学習サンプル数を増加させる実験を行った。ここではαは(0,0.2,0.9)の3種類を選択することにした。結果を図7に示す。

αについては学習初期及びサンプル数が増加した場合、α=0あるいは0.2をとり、それ以外はほとんどα=0.9を選択している。これは、学習の初期はリンクウェイトの修正方向が不安定であるため、αによる加速はオーバーシュートを起こし易いからであり、サンプル数が増加した場合も、エラー空間が変形するため過去のリンクウェイトの修正方向を引きずらない方がよいからである。

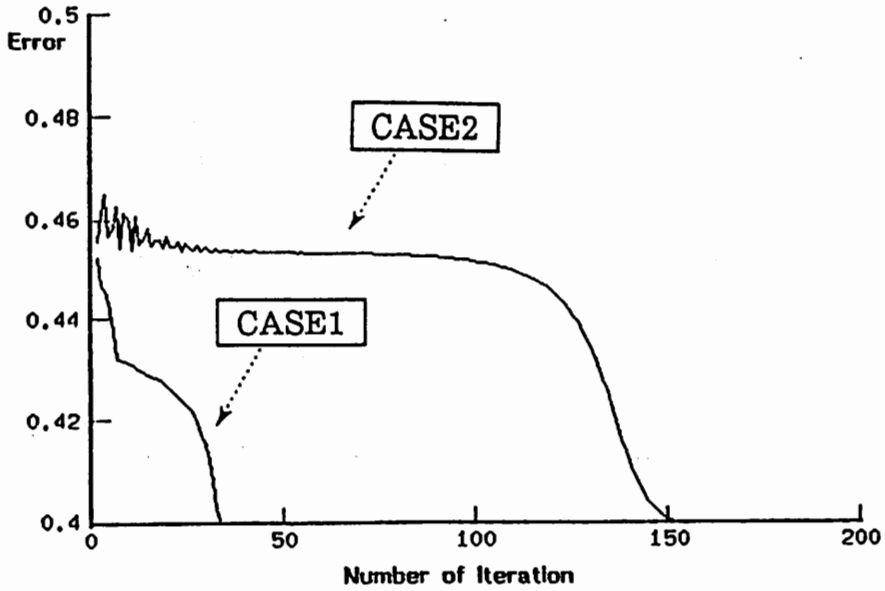
ηについてはサンプル数が増加するにしたがって減少している。すなわち、本方法ではサンプル数の大きさに対して自動的にηの値を正規化しているのがわかる。

以上、本方法を採用することにより、タスクの種類や学習サンプルの量に対して最適に近いパラメータ値を自動選択し、学習の進行状況に応じてパラメータを自動調整しており、結果として学習の効率化が図れている。

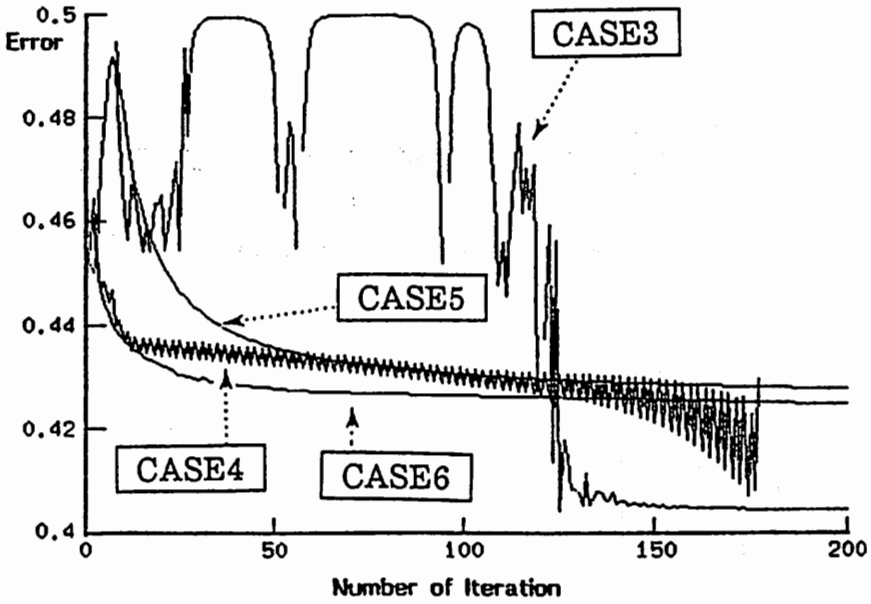
表ー1学習効率化法の効果確認実験結果

(単語列予測モデル、1センテンス学習)

	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5
η (ステップ幅)	(1/2, 1, 2)	0.1(fix)	0.4(fix)	0.4(fix)	0.1(fix)
α (モーメンタム)	(0.0 , 0.9)	0.9(fix)	0.9(fix)	0(fix)	0(fix)
繰り返し計算回数	35	153	200以上	178	200以上

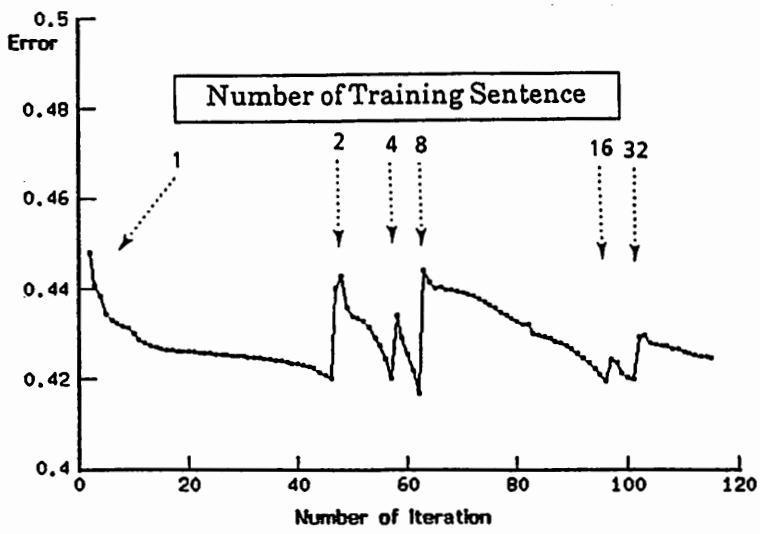


(a) Output Error vs. Number of Learning Iteration (CASE1,2)

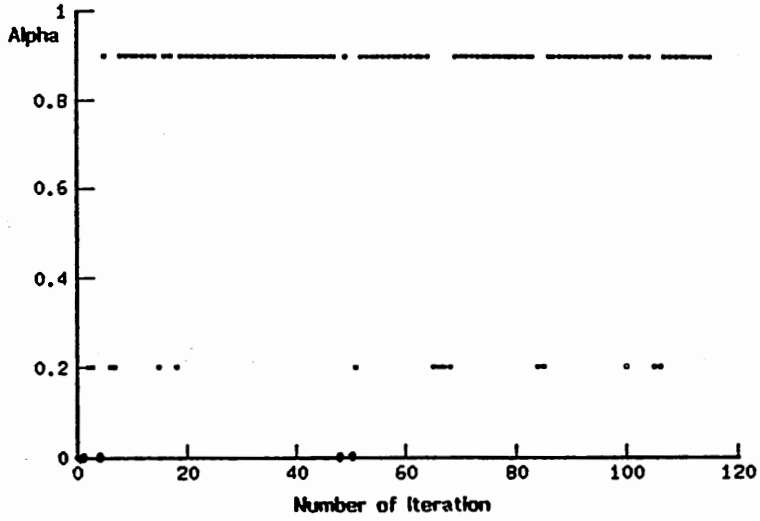


(b) Output Error vs. Number of Learning Iteration (CASE3,4,5,6)

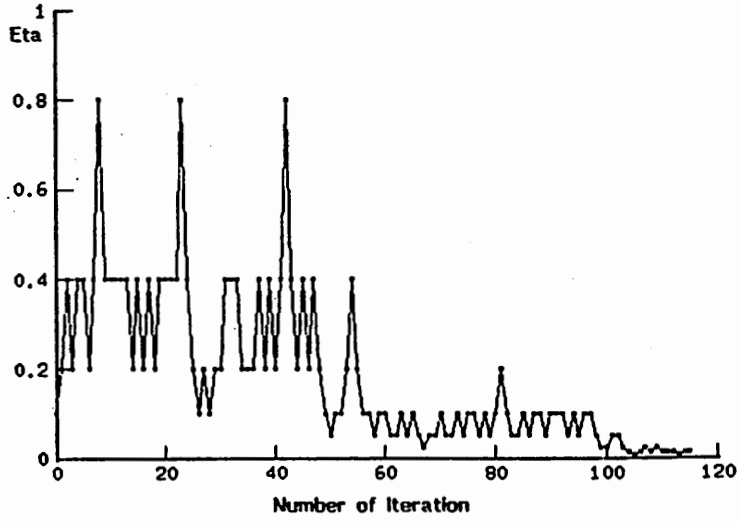
図6　学習効率化法の効果確認実験結果
(単語列予測モデル、1センテンス学習)



(a) Output Error vs. Number of Learning Iteration



(b) Momentum (Alpha) vs. Number of Learning Iteration



(c) Step Size (Eta) vs. Number of Learning Iteration

図7　学習効率化法の効果確認実験結果
(単語列予測モデル、トレーニングサンプル数増加学習)

5 DCP法の追加機能

5.1 DCP適用のインターバル化

前の実験では1回の繰り返し計算毎に全てのパラメータの組合せに対し計算を行ったため、η3種類、α2種類の場合、1回の繰り返し計算に6倍の時間を必要とした。しかし、実験結果から明らかなように、ある程度学習が進めば毎回パラメータを変更する必要はないので、変更のインターバルを大きくとることによりこの問題は解消される。

図8はその実験結果であり、繰り返し計算において、最初の10回は毎回DCPを行い、その後繰り返し計算4回のうち連続2回、8回のうち連続2回、16回のうち連続2回、以降32回のうち連続2回DCPを行った。連続2回にしたのはη,αの値が修正不足になるのを防ぐためである。また、出力誤差が前回の値より大きくなった場合や、αが小さい値を選択した場合もDCPを連続2回行うようにした。タスクはBigram ネットワークモデルでトレーニングサンプルは4文章(96語)である。

DCP適用のインターバル化により、出力誤差Ep<0.35に収束するCPU時間は139分から35分に大幅に短縮できた。計算回数は220回で、毎回DCP適用時と同程度である。

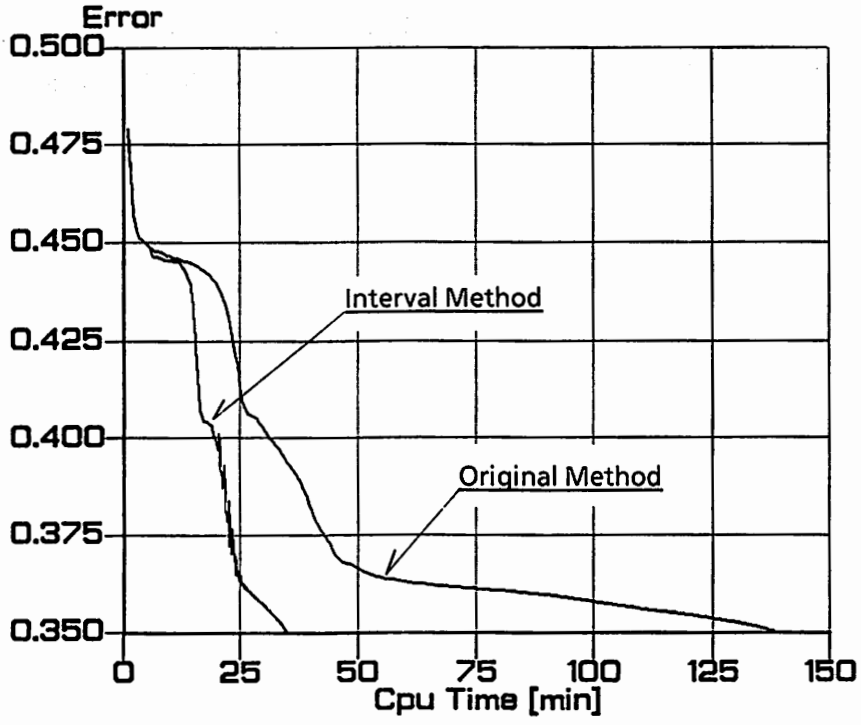


図8　DCP適用のインターバル化実験結果
(単語列予測タスク NETgram)

5.2 トレーニングサンプルの増加学習

TDNNによる音韻認識等のパターン認識の問題においては、最初から(重み係数がランダムな値から)大きなトレーニングサンプルで学習するのは非効率的である。例えば、bdg音韻認識タスクの場合、トレーニングサンプルをb,d,g 1サンプルずつのセットである程度学習した後、b,d,g のサンプルを同じ数だけ(出力誤差の閾値判定により)増加させていけば、学習が効率的に進む*。これはb,d,g 1サンプルずつのトレーニングセットでb,d,g.の大局的な特徴を学習させ、トレーニングセットのサイズを増加させることによって、詳細な特徴を学習させることを意味する。

ATRにおける別のバックプロパゲーション高速化プログラム DyNet[4] では複数のトレーニングセットを次々に学習させ(ひとつのトレーニングセット呈示の度に重み係数の修正を行う)、全てのトレーニングセットを一通り学習させたのち、トレーニングセットのサイズを大きくして(分割数を小さくして)同様の学習を繰り返すという分割学習方式をとっている。

DCP法では、DyNet と同じ方法をとると、毎回微妙に異なるトレーニングセットが呈示されるため、重み係数の修正方向が振動し、η,α の値が安定しなくなる可能性がある。そこで、DCP法ではトレーニングサンプルの増加学習を採用することにした。

図9は、bdg 音韻認識タスク(TDNN、全トレーニングサンプル99) を増加学習型DCP法で学習した結果である。最初から全トレーニングサンプル99を学習するDCP法に比べ、増加学習型DCP法の方が大幅に計算時間を削減しているのがわかる。

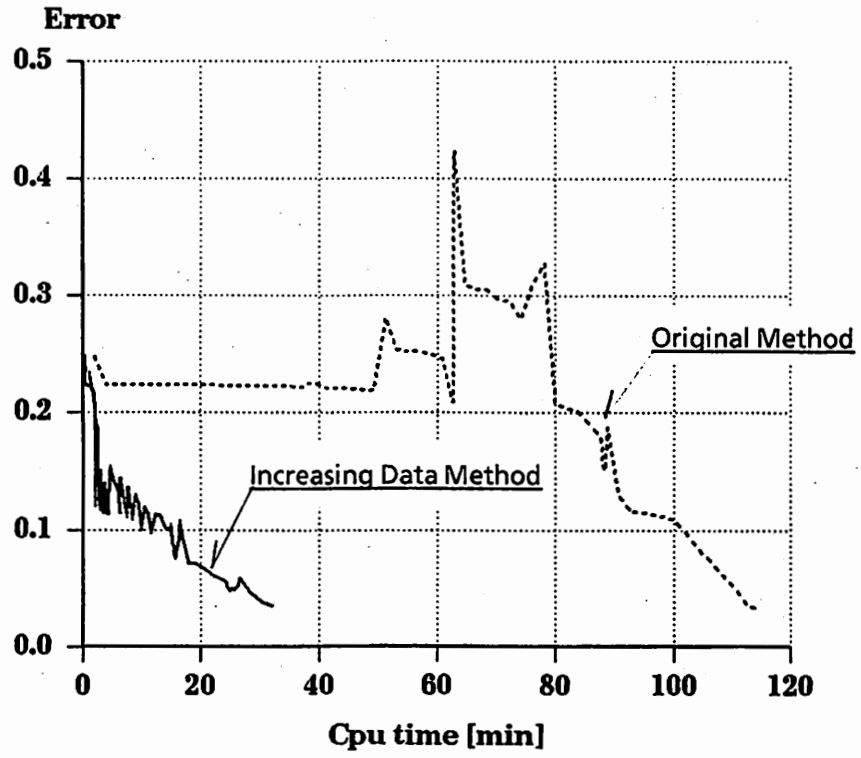


図9　トレーニングサンプル増加学習実験結果
(bdg音韻認識タスク、TDNN)

6 むすび

バックプロパゲーション法を用いた多層ニューラルネットの学習において、学習の程度を決定するパラメータを動的に変化させることにより、学習の効率化を図る方法を提案し、単語列予測タスク(NETgram)、bdg 音韻認識タスク(TDNN)に適用し、大幅な学習の効率化を達成した。本手法をUNIX上にインプリメントした学習プログラムDCP2の取扱説明、仕様を付録として掲載する。このプログラムは独立で動作するが、ATRで開発したニューラルネット ワークベンチシステム[8]と連動させることにより、ネットワークの作成、編集、学習状況のモニタリング、ネットワークの解析等が簡単に行え、より効率的なニューラルネット開発環境を構築することができる。

また、今回触れなかったが、ATRにおける別のバックプロパゲーション高速化プログラムDyNetとの比較については、別の機会に報告したい。

謝辞

研究の機会を与えて頂いた樽松社長、貴重な御助言、御討論頂いた音声情報処理研究室の皆様に感謝いたします。さらに、プログラムおよび、仕様書の作成に協力していただいた京都工芸繊維大学の石島氏に感謝いたします。

参考文献

1) D.E.Rumelhart, J.L.McClelland, "Parallel Distributed Processing : Explorations in the Micro Structure of Cognition", MIT Press, Vol. I, II, (1986)

2) D.E.Rumelhart, G.E.Hinton, R.J.Williams, "Learning Internal Representations by Back-Propagation Errors", Nature, 323, pp 533-536, (1986-10)

3) A.Waibel, T.Hanazawa, G.Hinton, K.Shikano, K.Lang, "Phoneme Recognition Using Time-Delay Neural Networks", ATR Technical Rep. TR-I-0006, (1987-10), あるいは, "Phoneme Recognition: Neural Networks vs. Hidden Markov Models", ICASSP'88, pp 107-110, (1988-03)

4) P.Haffner, A.Waibel, H.Sawai, K.Shikano, "Fast Back-Propagation Learning Methods for Neural Networks in Speech", ATR Technical Rep. TR-I-0058, (1988-11)

5) S.Tamura, A.Waibel, "Noise Reduction Using Connectionist Models", ICASSP'88, pp 553-556, (1988-04)

6) T.J.Sejnowski, C.R.Rosenberg, "NETtalk : A Parallel Network that Learns to Read Aloud", Technical Rep. JHU / EECS - 86 / 01, John Hopkins Univ. (1986-06)

7) 中村、鹿野、"英文テキストデータからのニューラルネットによる単語列予測モデルの検討", 電子情報通信学会技報,SP88-26, (1988-06)

8) 中村、田村、宮武、沢井、"ニューラルネット開発用ワークベンチシステム", ATRテクニカルレポート, TR-I-0113, (1989-9)

Appendix 1 ニューラルネット学習プログラム DCP2 ショートマニュアル

NAME

DCP2 - ニューラルネットワークシミュレーションプログラム

SYNOPSIS

dcp2 [option] [parameter] task

DESCRIPTION

DCP2 は DCP 法を使ったシミュレーションプログラム change をもとに、パラメータ変更タイミングのコントロールや、サンプルの分割やスキップなどの手法を取入れ、さらに高速化を計ったプログラムです。また、ユーザが自由に各種パラメータを変更し易いように、DyNet のようなパラメータファイルを使っています。

OPTION

-f < 区切り記号 >

通常、DCP2 は task/ サブディレクトリ以下の net などを読み込みますが、このオプションで区切り記号として ``.'' を指定すると、従来の task.net 式の名前のファイルも読めるようになっています。

PARAMETERS

DCP2 では、パラメータファイルのほかに、コマンドラインからもパラメータを指定できます。指定方法は、- [パラメータ名] [数値]... のようにします。コマンドラインで指定したパラメータは、パラメータファイルに優先します。

EXAMPLES

```
dcp2 task
dcp2 -f . task
dcp2 -ne 0 task
```

FILES

DCP2 は起動時にネットワークを記述した task/net と学習データを記述した task/sample を読み込みます。その次に、パラメータファイル task/para が存在すればこれを読みます。また、ウェイトデータファイル task/weight (圧縮版: DyNet スタイル) あるいは task/wght (非圧縮版) が存在すれば読み込んで、ウェイトデータの初期値とします。両方存在するときは task/weight が優先します。どちらも存在しないときは乱数で初期値を決めます。また、DCP2 は、一定間隔ごとに task/weight0, task/para0 というファイルにデータを書き出すので、これらを task/weight, task/para に mv すれば、学習を途中で中断しても再開することが可能です。さらに、task/monu, task/monw というファイルに学習状況を書き出すので、NNWB とリンクすることもできます。

Appendix 2 ニューラルネット学習プログラム DCP2仕様

A.2.1 概要

このプログラムは、DCP法を使ったシミュレーションプログラム change をもとに、サンプルの分割やスキップなどの手法を取入れ、さらに高速化を計ったシミュレーションプログラムです。
また、ユーザが初期値などのパラメータを変更しやすいように、DyNet のようなパラメータファイルを使っています。

A.2.2 基本的な使い方

まず、実行したいタスクを記述した net ファイル, sample ファイルをサブディレクトリ (以下 task) にいれます。つまり、task/net, task/sample というファイルを用意します。(ファイルの形式については後で述べます。)そのあと、

dcp2 task

とすると学習を始めます。

A.2.3 参照ファイル

dcp2 は起動時に task サブディレクトリの net, sample を読みます。これは学習する内容を書いたものですから、なかったら学習はできません。
つぎに、いつまで学習するか、といった学習に関する指令 (パラメータ) を書いてある para ファイルを読みに行きます。これはなくても構いません。ないときは適当に dcp2 が設定します。
そして、weight あるいは wght というファイルが存在すれば、これをウェイトの初期値として読み込みます。
dcp2 は現在のパラメータやウェイトデータなどを定期的にpara0, weight0 というファイルに書き出すので、これらを rename すれば、なんらかの理由で学習を中断した場合でも、学習を再開できます。

A.2.4 出力形式

dcp2 は標準出力にループ回数、CPU TIME、エラーなどを出力します。その形

式は、

```
DCP: combination = 6
6233 connections, 521 real weights
start from scratch
iter  time      alpha     eta       ep        reco      active learn
0     0.255000  0.500000  0.100000  0.263826  0.333333    9     9
1     0.321111  0.500000  0.200000  0.244875  0.333333    9    -9
```

となっています。
1行目はαとηの組合せ数、
2行目はコネクションの数と実際の(リンクを除いた)ウェイトの数、
3行目はウェイトの初期値を乱数で作ったか、あるいはファイルから読んだかを示します。
5行目からが実際の値で、左から、ループ回数,CPU TIME(単位は分),α,η,エラー,認識率,サンプル数,サンプル数,使ったサンプル数となっています。
使ったサンプル数がマイナスになることがありますが、これは、DCPでパラメータを変更したということを示すフラグで、実際に使ったサンプル数は正の値です。
なお、出力からループ回数とエラーの関係だけを抜きだしたいときは、

```
dcp2 task > outputfile
cat outputfile | tail +5 | awk '{print $1 " " $5}'
```

とすればいいでしょう。
また、dcp2をバックグラウンドで動かして、出力をoutputfileにリダイレクトしているときは、

```
tail +0f outputfile
```

とすれば、出力を観察することができます。

A.2.5 オプションとパラメータ

dcp2も他のUNIXのプログラムと同様にいろいろなオプション(パラメータ)があります。

さきに、学習に使うファイルは、サブディレクトリにまとめると書きましたが、従来との互換性を考えてtask.net形式のファイルも読めるようになっています。
これを指定するオプションは、-fで、

dcp2 -f . task

のようにします。
また、各種パラメータは通常、パラメータファイルで与えますが、コマンドラインからも指定できるようになっています。
指定の仕方は、パラメータの名前の前に"-"をつけ、たとえば、

dcp2 -ne 1 -ui 99 task

のようにします。

A.2.6 ネットワーク記述ファイル(net)

netファイルはネットワークの構造を記述したファイルです。
ファイルの形式はATR自動翻訳電話研究所で広く使われているWaibel's ASCII network formatです。
例えばxorなら

```
NETWORK FILE:
5 units
3 inputs
0 1 2
1 outputs
4
7 conns
 0 3
 1 3
 2 3
 0 4
 1 4
 2 4
```

3 4

のようになります。
1行目から見出し、
全ユニット数、
入力ユニットの数、
入力ユニットの番号、
出力ユニットの数、
出力ユニットの番号
の順となっています。
その後は、どのユニットからどのユニットへつながっているかを書きます。
ここで、リンク(つぎのコネクションと同じ値を使う)を使うときは行頭へ"*"を書きます。
また、ウェイトの値を固定したいときは、二つのユニット番号の次に"1"を書きます。これは行頭に"*"がないときにのみ有効です。("*"があるときは無視します。)

A.2.7 サンプルデータファイル(sample)

sampleファイルはネットワークの入力データと出力データを記述したファイルで、形式はWaibel's ASCII sample formatです。
xorなら、

SAMPLE FILE:
3 inputs
1 outputs
4 patterns
1 0 0　　0
1 1 0　　1
1 0 1　　1
1 1 1　　0

のようになります。
これも、一行目から、見出し、入力のユニット数、出力ユニットの数、サンプルデータの数をかいて、5行目から、入力値と出力値を並べて書きます。

A.2.8 パラメータファイル

パラメータファイルは、行頭から

[2文字で表したパラメータのなまえ][設定する値][コメント]

の順に書きます。
ただし、コメントは数値ではじめないようにしてください。

A.2.8.1 いつまで学習させるか

dcp2 に学習期間を指定する方法は2つあり、
ひとつは、ループ回数を指定する方法(ne)、
もう一つは、エラーの値を指定する方法(on)です。
もし、どちらか一方だけの条件を使いたいとき(例えば、エラーの値はどうでもいいが、10回だけ学習させたいときなど)は、使わない条件の値を 0 にしてください。

A.2.8.2 モーメンタムαとステップサイズηの初期値

初期値はそれぞれ ai と ei で指定し、ai, ei の後ろに初期値を小さい方から書き並べます。
また、ei の指定で倍率も指定できます。
たとえば、初期値を 0.1 にして、倍率を 1/2, 1, 2 にするときは

ei 0.05 0.1 0.2

のようにします。
もちろん、倍率は可変で

ei .0707106781 0.1 0.2

のような指定もできます。
また、ηが大きくなりすぎたり、小さくなりすぎるのを防ぐため、上限値

と下限値を em と en で指定できます。

A.2.8.3 DCP インターバルの指定

dcp2 では、高速化のために、最初の数回は毎回パラメータを変更し、そのあとはパラメータの変更頻度を小さくしていきます。つまり、変更してから次に変更するまでのインターバルをだんだん大きくしていくのです。
この最初の数回というのを ci で指定します。
そして、大きくしていく倍率を ce で指定し、
インターバルの最大値を cm で指定します。
また、1インターバル中に一度しか変更しないと、不適切な値を選んでしまい、かえってエラーが大きくなったりすることがあるので、
1インターバル中で何度変更するかを cn で指定できるようになっています。
また、小さい方のαを選んだとき、エラーの値が前回よりも大きくなったときに、さらにもう一度パラメータを変更するかどうかをそれぞれ ca と cr で選択できます。

A.2.8.4 サンプルのスキップ

あるサンプルに対するエラーが十分小さくなったときは、そのサンプルをしばらく学習しないようにできます。
この、"十分小さくなった" という値を om で指定します。
そして、"しばらく" というのを oe で指定します。
つまり、あるサンプルに対するエラーに oe を加えていった値が、om より小さい期間は学習をしない、という仕組みになっています。

A.2.8.5 サンプルの分割

もし、サンプルの中のちいさなサブセットが、そのサンプル全体の特徴を表現しているなら、そのサブセットだけを使ってある程度学習させておき、そのあとで全体を使って学習させたり、あるいは、サンプルをいくつかのグループに分解できるなら、学習するサンプルの数をそのグループに合わせて増やしたりすることができます。
サンプル数の初期値を ui で、

増やし方を ue で、
サンプル数の最大値を um でそれぞれ指定します。
また、サンプル数は、エラーが uc で指定した値よりも小さくなったら増やします。

A.2.8.6 その他

dcp2 では、再スタートのためのデータを weight0, para0 というファイルに一定の期間毎に書き込みます。
この期間は in で指定します。
そして、nnwb (ニューラルネットワークワークベンチ) とのデータのやりとりをするファイル monw, monu も一定の期間毎に書き込んでいます。
これは、iw と iu で指定します。
また、出力ユニットに sigmoid 関数を使わないかどうかを ol で指定します。

A.2.8.7 パラメータファイルの例

以上のまとめとして、パラメータファイルの例を書いておきます。
なお、無効な指定をすると無視されるか、ディフォルトに設定し直されます。

```
#
#パラメータファイルの例
#
#'#'ではじまる行はコメントです。
#見やすくするために空行も使えます。

#'['と']'でくくってあるのは指定がなかったときの値(ディフォルト)です。

#いつまで学習するか
ne 100      学習回数は100回まで(0なら無制限)  [100]
on 1e-5     エラーが1e-5未満になるまで        [1e-5]

#alphaとeta
ai 0.1 0.9          alphaの変更選択値          [0.1 0.9]
ei 0.005 0.01 0.02  etaの変更選択値の初期値     [0.005 0.01 0.02]
em 10               eta上限値                  [0: 制限無し]
en 0.00001          eta下限値                  [0: 制限無し]
```

```
                    # em, en は 0 を指定すると "制限無し" になります

# DCPのインターバル
ci 10      最初の ci 回は毎回変更し、                [10]
ce 2       そのあとは、ce, ce^2, ce^3,... 回に cn 回変更し、  [2]
           # ce を 0 にすると alpha, eta の変更をしません。
           # 逆に 1 にすると毎回変更します。
cm 32      最終的には cm 回に連続 cn 回変更          [32]
cn 2       1インターバル中で連続 cn 回変更           [2]
cr 1.0     エラーが前回の cr 倍以上なら変更 (0 なら無視)   [1.0]
ca 1       alpha が小さいときに変更(1) / 変更しない(0)    [1]

# サンプルのスキップ
om 1e-6    一つの学習サンプルのエラーがこの値より小さいときは
           # そのサンプルについてのみ学習しない      [on の 1/10]
oe 2e-7    上記学習をしない間を決めるパラメータ      [om の 1/5]
           # om を 0 にするとサンプルスキップをしません。

# サンプルの分割
ui 4       サンプル数の初期値                [0: 全サンプル数]
ue 4       増分                              [0]
um 16      最大値                            [0: 全サンプル数]
uc 1e-4    エラーが uc より小さくなったらサンプルを増やす [on の 10 倍]
           # ui あるいは um を 0 にすると分割をしません。

# その他
in 10      weight0, para0 書き込みインターバル      [10]
iu 1000    monu 書き込みインターバル
           # [-1: サンプル数 * alpha の数 * eta の数 + 1]
iw 10      monw 書き込みインターバル                [10]
           # i ? を 0 にすると終了時にのみ書き込みます。
ol 0       出力を sigmoid に通す(0) / 通さない(1)   [0]
rs 0       乱数の種                          [0]
wi 0.3     乱数でウェイトデータを決めるときの幅      [0.3]
```

A.2.9 関数の説明

::: DCP2/src/ans.h :::

```
swap(type, a, b)
  型指定子 type;
  type   a, b;
```

type 型の変数 a と b を入れ換えます。(これはマクロです。)

使用例: int a,b; .... swap(int, a, b);

```
    char *s1, *s2; .... swap(char *, s1, s2);
```

::: DCP2/src/backprop.c :::

```
backprop(num, conns, parm, weight, unit, ae, samplein, sampleout, file)
  numbers  *num;
  c_info   *conns;
  parameter *parm;
  w_info   **weight;
  u_info   **unit;
  alpha_eta **ae;
  real     **samplein;
  real     **sampleout;
  fname    *file;
```

バックプロパゲーションのメインです。

おおまかなフローは

0) 内部で使用する変数の初期化、メモリ割り当て

1) ウェイトファイルがあれば、それを読んでフォワード計算
　なければ乱数を使った初期値でフォワード計算

2) パラメータ on, ne で指定した終了条件なら終了

2-1) パラメータを更新するタイミングなら

2-1-1) αとηを設定

2-1-2) ウェイト変更

2-1-3) 学習

をαとηの組合せの数だけ繰り返し、一番エラーの少ないものを選択

2-2) 変更のタイミングでなければ、ウェイト変更と学習を1回だけ行う

2-3) 必要に応じて状況の表示、ファイルへの出力

3) 2) へ戻る

となっています。

```
int is_dcp(iter, parm, ae, pre_ch, ch_int, min_es, last_es)
  int iter;            イタレーション
  parameter *parm;
  alpha_eta *ae;
  int *pre_ch;         前回変更したイタレーション
  int *ch_int;         変更のインターバル
  double min_es, last_es;  現在のエラー値と前回のエラー値
```

引数をもとにして、dcp パラメータの更新をするかどうかを判断します。
更新するのは以下のような時です。

1) iteration が ci で指定した値より小さいとき
2) 小さい方のαを選択したとき
3) エラーが前回よりも大きかったとき
4) ce, cm, cn で指定された更新タイミングの時

```
new_alpha_eta(parm, i, ae, pre_ae)
  parameter *parm;
  int    i;
  alpha_eta *ae;    これに更新されたαとηの組合せが入る
  alpha_eta *pre_ae; 前回のαとηの組合せ
```

αとηの更新を行います。

```
double learn(num, parm, wght, unit, as, s_in, s_out, ep, ep0, ns, reco, file)
  numbers  *num;
  parameter *parm;
  w_info   wght[];
  u_info   unit[];
  int    as;    学習に使用すべきサンプルの数
  real   **s_in;
  real   **s_out;
  real   ep[];  各サンプルに対するエラー値
  real   ep0[];  前回の ep[]
  int    *ns;   学習に使用されたサンプルの数を返す
  real   *reco;  認識率を返す
```

```
  fname   *file;
```

1イタレーション分の学習をします。

```
report(iter, cputime, ep, ae, as, ns, reco)
  int iter;
  double cputime;
  double ep;
  alpha__eta *ae;
  int as;        学習に使用すべきサンプルの数
  int ns;        使用されたサンプルの数
  real reco;
```

標準出力に iteration, error などを出力します。

::: DCP2/src/build.c :::

```
net__build(num, conns, weight, unit)
  numbers *num;
  c__info conns[];
  w__info weight[];
  u__info unit[];
```

コネクション情報 conns[] をもとにしてネットワークを構成します。

```
init__dcp(num, conns, weight, unit, ae)
  numbers *num;
  c__info conns[];
  w__info ***weight;
  u__info ***unit;
  alpha__eta ***ae;
```

αとηの組合せの数だけ変数を確保します。

::: DCP2/src/file.c :::

```
readnet(netfile, num, conns)
  char *netfile;
  numbers *num;
  c__info **conns;
```

netファイルを読んで、コネクション情報をconnsにセットします。
conns[]のメモリ割当は、ここで行います。

```
readsample(samplefile, num, samplein, sampleout)
  char *samplefile;
  numbers *num;
  real ***samplein, ***sampleout;
```

sampleファイルを読んで、samplein, sampleoutのメモリを割り当て、
サンプルデータを格納します。

```
writeweight(weightfile, num, weight)
  char *weightfile;
  numbers *num;
  w_info weight[];
```

ウェイトデータをweightファイルに(圧縮形式で)書き込みます。

```
int readweight(weightfile, num, weight)
  char   *weightfile;
  numbers *num;
  w_info  weight[];
```

weightファイル(圧縮形式)を読みます。
ファイルがみつからなかったらNOを返します。

```
readwght(wghtfile, num, conns, weight)
  char *wghtfile;
  numbers *num;
  c_info  conns[];
  w_info  weight[];
```

wghtファイル(非圧縮形式)を読みます。
ファイルがみつからなかったらNOを返します。

```
readln(fp)
  FILE *fp;
```

行末まで読み飛ばします。

```
int skipbl(fp)
  FILE *fp;
```

ブランクとタブを読み飛ばします。

::: DCP2/src/main.c :::

```
main(argc, argv)
  int argc;
  char *argv[];
```

main です。

```
int option(argc, p, argv, parm)
  int argc;
  int p;
  char *argv[];
  parameter *parm;
```

コマンドラインのパラメータを調べて、setpara() を使って設定します。

::: DCP2/src/monitor.c :::

```
mon_header(fp, num, es)
  FILE    *fp;
  numbers  *num;
  double   es;
```

NNWB 用のモニターファイルのヘッダを書きます。

```
mon_footer(fp)
  FILE *fp;
```

同じくフッタを書きます。

```
mon_set(iter, ae0, reco)
  int iter;
  alpha_eta *ae0;
  double reco;
```

モニターファイル用の数値をセットします。

```
mon_wght(monfile, num, conns, weight, iter, ep, reco, ae0)
  char    *monfile;
  numbers  *num;
  c_info   conns[];
```

```
  w_info  weight[];
  int     iter;
  double  ep, reco;
  alpha_eta *ae0;
```

monw ファイルを書きます。

```
mon_unit(monfile, num, unit, sampleout, ep)
  char    *monfile;
  numbers *num;
  u_info  unit[];
  real    sampleout[];
  double  ep;
```

monu ファイルを書きます。

::: DCP2/src/para.c :::

```
initparm1(parm)
  parameter *parm;
```

パラメータの初期値を設定します。

```
initparm2(num, parm)
  numbers *num;
  parameter *parm;
```

パラメータのうち、om, oe, ui, um, uc, iu などがパラメータファイルの指定になかったときに適当な値に設定します。

```
readpara(parafile, parm)
  char    *parafile;
  parameter *parm;
```

パラメータファイルを読んで、パラメータを設定します。

```
int setpara(n, arg, parm)
  int     n;      arg[] の数
  char    *arg[]; パラメータと数値をいれた文字列の配列
  parameter *parm;
```

パラメータの設定を行います。該当するパラメータがなければ NO を返します。

```
write__para(parafile, parm, ae, iteration, cputime, act__sample)
  char    *parafile;
  parameter *parm;
  alpha__eta *ae;
  int     iteration;
  double  cputime;
  int     act__sample;
```

再起動用のパラメータファイルを書きます。

::: DCP2/src/pass.c :::

```
set__in(num, unit, samplein)
  numbers *num;
  u__info *unit;
  real   *samplein;
```

入力ユニットに学習データをセットします。

```
forward(num, unit, parm)
  numbers  *num;
  u__info  unit[];
  parameter *parm;
```

フォワード計算を行います。

```
double error(num, unit, sampleout, reco)
  numbers *num;
  u__info unit[];
  real   sampleout[];
  int   *reco;   認識していたら1を返します。
```

出力ユニットの2乗誤差の和を返します。

```
backward(num, unit, parm)
  numbers  *num;
  u__info  unit[];
  parameter *parm;
```

バックワード(δp)の計算をします。

```
new__dedw(num, weight)
  numbers *num;
```

```
  w_info weight[];
```

dE/dw を計算します。

```
new_weight(num, weight, pre_w, ae)
  numbers  *num;
  w_info   weight[]; 更新されたウェイト
  w_info   pre_w[]; 前回のウェイト
  alpha_eta *ae;
```

ウェイトの更新をします。

::: DCP2/src/random.c :::

```
double rnd()
```

0.0 以上 1.0 未満の乱数を発生します。

::: DCP2/src/sigmoid.c :::

```
double sigmoid(x)
  double x;
```

sigmoid 関数です。

```
double dsig(x)
  double x;
```

sigmoid の導関数です。

```
init_f()
```

テーブルサーチのテーブルを初期化します。

```
double t_sigmoid(x)
  double x;
```

sigmoid のテーブルサーチ版です。
コンパイル時に make clean; make CFLAGS=-DNO_TAB とすると
テーブルサーチを使いません。

```
double t_dsig(x)
  double x;
```

sigmoid の導関数のテーブルサーチ版です。

::: DCP2/src/timer.c :::

double timer()
プロセスの使用時間 (単位は分) + set__offset() で指定した値を返します。

set__offset(x)
  double x;
プロセスの使用時間のオフセットを設定します。
再起動されたときに使用します。

つぎに、関数の構造図を示します。

```
main
    readnet
        readln
    readsample
        readln
    initparm1
    initparm2
    readpara
        skipbl
        setpara
        readln
    option
        setpara
    init__f
    init__dcp
        net__build
    backprop
        mon__set
        set__offset
        rnd
        readweight
        readwght
        learn
            set__in
```

```
    forward
        f{ sigmoid or t_sigmoid }
    error
    backward
        dfdx{ dsig or t_dsig }
    new_dedw
    mon_unit
        mon_header
        mon_footer
swap
timer
report
    getformat
is_dcp
new_alpha_eta
new_weight
write_weight
write_para
mon_wght
    mon_header
    mon_footer
mon_unit
    mon_header
    mon_footer
```

A.2.10 構造体によるネットワーク

DCP2 ではネットワークの構造を以下の構造体(の配列)で持っています。

```
typedef struct {
  real net;        ユニットの入力
  real o;          ユニットの出力
  real error;      教師信号との差
  real delta;      δp
  int fan_in;      前段ユニットの数
  real **pre_o;    前段ユニットの出力をさすポインタの配列
  real **pre_w;    入力側ウェイトの値をさすポインタの配列
  u_info **pre_unit; 前段ユニットをさすポインタの配列
  int fan_out;     後段ユニットの数
  real **next_delta; 後段ユニットの δp をさすポインタの配列
  real **next_w;   出力側ウェイトの値をさすポインタの配列
  u_info **next_unit; 後段ユニットをさすポインタの配列
} u_info;

typedef struct {
  real w;          ウェイトの値
  real dw;         △w
  real dedw;       -dE/dw
  int n_link;      リンク数
  real **delta;    出力側ユニットの δp をさすポインタの配列
  u_info **next_unit; 出力側ユニットをさすポインタの配列
  real **o;        入力側ユニットの出力値をさすポインタの配列
  u_info **pre_unit; 入力側ユニットをさすポインタの配列
  char is_fixed;   ウェイト値を変更しないかどうかを決めるフラグ
} w_info;

typedef struct {
  int from;        入力側ユニットの番号
  int to;          出力側ユニットの番号
  int weight;      ウェイトの番号
  char is_linked;  リンクフラグ
  char is_fixed;   ウェイト値を変更しないかどうかを決めるフラグ
```

}c_info;

例えば、10番目のユニットの入力に 0, 4, 5, 6 の4つのユニットがつながっていて
出力に 12, 13, 14 の3つのユニットがつながっているとき、
unit[10] の構造体の要素は

unit[10].fan_in = 4
unit[10].pre_o[0] -> unit[0].o　　-> はポインタが指している
unit[10].pre_o[1] -> unit[4].o　　という意味です。
unit[10].pre_o[2] -> unit[5].o
unit[10].pre_o[3] -> unit[6].o
unit[10].fan_out = 3
unit[10].next_delta[0] -> unit[12].delta
unit[10].next_delta[1] -> unit[13].delta
unit[10].next_delta[2] -> unit[14].delta

のようになります。
さらに、入力側のウェイトの番号が 9, 22, 25, 28 で、
出力側が 40, 41, 42 なら、

unit[10].pre_w[0] -> weight[9].w
unit[10].pre_w[1] -> weight[22].w
unit[10].pre_w[2] -> weight[25].w
unit[10].pre_w[3] -> weight[28].w
unit[10].next_w[0] -> weight[40].w
unit[10].next_w[1] -> weight[41].w
unit[10].next_w[2] -> weight[42].w

のようになります。
そして、forward/backward の計算は simul や change と違って
ユニットに着目して行われます。
¥ (詳しくは pass.c を見てください。)
また、メモリの節約と高速化のために
ウェイトデータは実際の (リンクを除いた) ウェイト分しか使いません。
例えば、net ファイルが

NETWORK FILE:
5 units
3 inputs
0 1 2
1 outputs
4
7 conns
0 3
0 4
*1 3
2 4
*1 4
2 3
3 4

となっていたら、コネクションの数は7ですが、
ウェイトの数は5になります。
さらに、weight[] は

weight[0].n_link = 1
weight[0].delta[0] -> unit[3].delta
weight[0].o[0] -> unit[0].o

中略

weight[2].n_link = 2
weight[2].delta[0] -> unit[3].delta
weight[2].delta[1] -> unit[4].delta
weight[2].o[0] -> unit[1].o
weight[2].o[1] -> unit[2].o

中略

weight[4].n_link = 1
weight[4].delta[0] -> unit[4].delta
weight[4].o[0] -> unit[3].o

となります。
これらの構造体は、build.c の net__build() で設定されます。