$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) \tag{5.43}$$

とする．この更新はデータを順番に繰り返すか，ランダムに復元抽出して繰り返す．もちろん，いくつかのデータ点を1まとめにした中間的なシナリオもある．

バッチ手法と比べ，オンライン手法の利点の1つは，データの冗長度をはるかに効率的に扱うことができることである．このことを理解するため，あるデータ集合を取り，すべてのデータ点を複製してサイズを倍にした極端な例を考えよう．これは誤差関数を単に2倍しただけであり，もとの誤差関数を用いることと等価であることに注意しよう．バッチ手法ではバッチ誤差関数の勾配を評価するのに2倍の計算量が必要となるが，一方，オンライン手法では影響がない．オンライン勾配降下法のもう1つの性質として，局所的極小値を回避できる可能性があることが挙げられる．というのは，すべてのデータ集合についての誤差関数による停留点は，個々のデータ点に対しては一般に停留点ではないからである．

非線形最適化アルゴリズムとニューラルネットワークの訓練へのその実用的な応用は，Bishop and Nabney (2008) において詳細に議論されている．

5.3　誤差逆伝播

本節での目標は，フィードフォワードニューラルネットワークについて，誤差関数 $E(\mathbf{w})$ の勾配を効率良く評価するテクニックを見つけることである．後に見るように，これは順向きと逆向きの交互に情報がネットワークを流れる局所的なメッセージパッシングスキームを用いて実現でき，**誤差逆伝播** (error backpropagation または単に backprop) として知られている．

ここで，逆伝播という語は，ニューラルコンピューティングの文献では，さまざまな異なる事柄を意味するのに使われていることに注意が必要である．例えば，多層パーセプトロンの構造はしばしば逆伝播ネットワークと呼ばれる．逆伝播という語は，勾配降下法を二乗和誤差関数に応用した多層パーセプトロンの訓練を表すのにも用いられる．用語を明確にするには，訓練過程の本質をより注意深く考えるとよい．ほとんどの訓練アルゴリズムには誤差関数を最小化するための反復手続きがあり，一連のステップにおいて重みを調整する．各ステップは2つの異なるステージに分けられる．最初のステージでは誤差関数の重みに関する微分を評価しなければならない．後に見るように，逆伝播テクニックが特に貢献しているのは，そのような微分の評価において計算論的に効率的な方法を提供している点にある．誤差がネットワークを逆方向に伝播するのはこのステージなので，逆伝播という語は微分の評価を表すのによく使われる．2番目のステージでは，微分を用いて重みの調整量が計算される．そのための

には，勾配降下法が関係している．この2つのステージは別々であることを理解することが重要である．したがって，最初のステージ，すなわち微分評価のためのネットワーク上の誤差の逆伝播は，多層パーセプトロンだけでなく，多くの他の種類のネットワークにも応用可能である．これはさらに，簡単な二乗和以外の誤差関数にも応用可能であり，さらにはヤコビ行列やヘッセ行列のような他の微分の評価にも用いることができる．このことは本章の後ろの方で述べる．同様に，計算された微分を用いて重みを調節する2番目のステージは，その多くが単純な勾配降下法よりも実質上より強力なさまざまな最適化のスキームと組み合わせることができる．

5.3.1　誤差関数微分の評価

それでは一般のネットワーク，すなわち任意のフィードフォワード構造と任意の微分可能な非線形活性化関数，そして広いクラスの誤差関数を持つネットワークについての逆伝播アルゴリズムを導出しよう．そして得られた式を，シグモイド隠れユニット1層と二乗和誤差を持つ単純な多層ネットワーク構造を用いて図示しよう．

独立同分布のデータ集合について最尤推定で定義される場合など，実用上興味ある多くの誤差関数は，訓練集合の各データに対応する誤差項の和，すなわち

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{5.44}$$

として表される．ここでは，そのような誤差関数の1つの項に対する勾配 $\nabla E_n(\mathbf{w})$ を評価する問題を考えよう．逐次的最適化にはこれを直接用いればよく，バッチ手法の場合には訓練集合全体についての和をとればよい．

まず，出力 y_k が入力変数 x_i の線形和

$$y_k = \sum_i w_{ki} x_i \tag{5.45}$$

という単純な線形モデルで，ある特定の入力パターン n に対する誤差関数が

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{5.46}$$

という形を取る場合を考えよう．ただし，$y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$ である．この誤差関数の重み w_{ji} に関する勾配は

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{5.47}$$

で与えられ，これは，リンク w_{ji} の出力側の「誤差信号」$y_{nj} - t_{nj}$ とリンクの入力側の変数 x_{ni} の積という「局所的な」計算であると解釈することができる．4.3.2節で

類似の式を導出し，そしてソフトマックス活性化関数およびそれに対応する交差エントロピー誤差関数の場合も同様であった．ここでは，この単純な結果を，より複雑な多層フィードフォワードネットワークに拡張する方法を見よう．

一般のフィードフォワードネットワークでは，それぞれのユニットは

$$a_j = \sum_i w_{ji} z_i \tag{5.48}$$

の形の入力の重み付き和を計算する．ここで，z_i はユニット j に結合があるユニットの出力，すなわち（ユニット j への）入力であり，w_{ji} はその結合の重みを表す．5.1 節で見たように，出力が $+1$ に固定された余分なユニット，すなわち入力を導入することで，バイアスはこの和に含めることができる．したがってここではバイアスを明示的に扱う必要はない．(5.48) の和は非線形活性化関数 $h(\cdot)$ によって変換され，ユニット j の出力 z_j が

$$z_j = h(a_j) \tag{5.49}$$

の形で与えられる．ここで，(5.48) の和における変数 z_i のいくつかは入力であり得るし，同様に，(5.49) のユニット j は出力であり得ることに注意する．

訓練集合のそれぞれのパターンに対し，対応する入力ベクトルがネットワークに与えられ，(5.48) と (5.49) を順番に適用することによりネットワークのすべての隠れユニットと出力ユニットの出力が計算されているとしよう．この過程は情報がネットワークを順向きに流れるとみなせるので，しばしば**順伝播** (forward propagation) と呼ばれる．

それでは E_n の重み w_{ji} に関する微分の評価を考えよう．さまざまなユニットの出力は特定の入力パターン n に依存する．しかしながら，表記を単純にするため，以下ではネットワークの変数から添え字 n を省略する．まず，E_n はユニット j への入力和 a_j を通してのみ重み w_{ji} に依存することに着目する．これにより，偏微分の連鎖法則を適用でき，

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{5.50}$$

が与えられる．ここで，便利な記法

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{5.51}$$

を導入しよう．ただし，δ はしばしば**誤差**と呼ばれる（その理由はすぐわかる）．(5.48) を用いると

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{5.52}$$

と書くことができ，(5.51) と (5.52) を (5.50) に代入すると

が得られる．式 (5.53) により，ある重みの出力側のユニットの δ の値と重みの入力側のユニットの z の値（バイアスなら $z = 1$）を掛け合わせるだけで，必要な微分が得られることがわかる．これは，本節の最初に考えた，単純な線形モデルと同じ形を取ることに注意しよう．したがって微分を評価するには，ネットワークの各隠れユニットおよび出力ユニットの δ_j の値を計算し，(5.53) を適用するだけでよい．

すでに見てきたように，正準連結関数を活性化関数に用いた出力ユニットでは

$$\delta_k = y_k - t_k \tag{5.54}$$

である．隠れユニットの δ を評価するには，再度，偏微分の連鎖法則を利用すればよく，

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{5.55}$$

となる．ただし，この和はユニット j に結合があるすべてのユニット k について行う．ユニットと重みの配置については図 5.7 に示されている．ここで，k とラベル付けされたユニットは，他の隠れユニットや出力ユニットを含み得る点に注意する．(5.55) を書き下す際には，a_j の変化は a_k の変化を通してだけ，誤差関数を変化させるという事実を利用している．もし (5.51) の δ の定義を (5.55) に代入し，(5.48) と (5.49) を用いれば，

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k \tag{5.56}$$

という**逆伝播**公式が得られる．この式は図 5.7 にあるように，ある特定の隠れユニットの δ の値はネットワークの上流のユニットから δ を逆向きに伝播させて得られることを意味している．ここで，(5.56) の和は w_{kj} の最初の添え字（ネットワーク上の情報の逆伝播に相当）について取ることに注意する．これに対し，順伝播の方程式を表す (5.10) では，和は 2 番目の添え字について取る．出力ユニットの δ の値はすでにわかっているので，(5.56) を再帰的に適用することにより，ネットワークの構造によらず，フィードフォワードネットワークのすべての隠れユニットの δ が評価できる．

以上をまとめると，逆伝播の手続きは以下のように書ける．

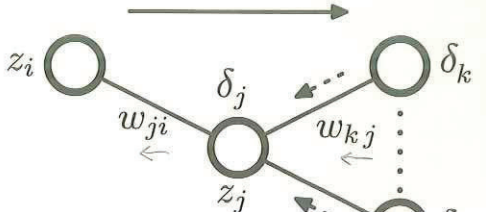

図 5.7　隠れユニット j の δ_j をこのユニットに結合を持つ k 個のユニットの δ を逆伝播して計算する様子．実線の矢印は順伝播での情報の流れの向きを表し，破線の矢印は誤差情報の逆伝播を表している．

✤ 誤差逆伝播 ✤

1. 入力ベクトル $\mathbf{x}_n$ をネットワークに入れ，(5.48) と (5.49) を用いてネットワーク上を順伝播させ，すべての隠れユニットと出力ユニットの出力を求める．
2. (5.54) を用いてすべての出力ユニットの δ_k を評価する．
3. (5.56) を用いて δ を逆伝播させ，ネットワークのすべての隠れユニットの δ_j を得る．
4. (5.53) を用いて必要な微分を評価する．

バッチ手法については，全体の誤差 E の微分は，上のステップを訓練集合のそれぞれのパターンについて繰り返し，すべてのパターンについて和をとることで得られる．すなわち，

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \tag{5.57}$$

となる．上の導出では，ネットワーク内のそれぞれの隠れユニットや出力ユニットが同じ活性化関数 $h(\cdot)$ を持つことを暗に仮定していた．しかしながら，この導出を異なるユニットが個々の活性化関数を持つ場合に拡張するのは簡単であり，単にどのユニットがどんな $h(\cdot)$ を持つかに注意すればよい．

5.3.2　単純な例

逆伝播手続きについての上の導出は，どんな形の誤差関数，活性化関数，ネットワーク構造についてもあてはまる一般的なものであった．このアルゴリズムの応用を示すため，特定の例について考えてみよう．簡潔さと実用上の重要さを考慮し，多くの文献でニューラルネットワークの応用例に利用されているモデルを用いる．具体的には，図 5.1 の構造を持つ 2 層ネットワークで，二乗和の誤差関数を持ち，出力ユニットは線形活性化関数 $y_k = a_k$，一方，隠れユニットは

$$h(a) \equiv \tanh(a) \tag{5.58}$$

で与えられるシグモイド活性化関数を持つものを考える．ここで

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{5.59}$$

である．

この関数はその微分が特に単純な形で書けるという便利な性質を持つ．すなわち

誤差は (5.61)

ユニッ (5.62) (5.63) (5.64) (5.65) 5.66) 5.67)

neuroBack.txt

```
#number of hidden unit
s <- 30

#learning rate parameter
l <- 0.05

#count of loop
MAX_EPOCH <- 1000

#read traning data
digit_data <- read.csv("train/mnist_train_all.txt",header=F)

##initialize weight parameter
#weight parameter between input and hidden units
w1 <- matrix(runif(s*length(digit_data), -1, 1), s, length(digit_data))
#weight parameter between hidden units and output
w2 <- matrix(runif(10*(s+1), -1, 1), 10, s+1)

#definition of function
logsumexp <- function (x) {
  m <- max(x)
  m + log(sum(exp(x-m)))
}

softmax <- function (a) {
  sapply(a, function (x) { exp(x-logsumexp(a)) })
}

neuro_func <- function (input) {
  a1 <- w1 %*% c(1, input)
  z1 <- tanh(a1)
  a2 <- w2 %*% c(1, z1)
  z2 <- softmax(a2)
  return(z2)
}

##train the neural network
for (k in 1:MAX_EPOCH) {

  #sample 1000 training data
  sample_index = sample(1:nrow(digit_data), 1000)

  for (i in sample_index) {

    tmp <- digit_data[i,]

    #target data
    t <- rep(0, 10)
    t[as.integer(tmp[1]+1)] <- 1

    input <- t(tmp[2:length(tmp)])

    #forward propagation
    a1 <- w1 %*% c(1, input)
    z1 <- tanh(a1)
    a2 <- w2 %*% c(1, z1)
    z2 <- softmax(a2)

    #back propagation
    d2 <- z2 - t
    w2 <<- w2 - l * d2 %*% t(c(1, z1))
    d1 <- d2 %*% w2[,2:(s+1)] * (1-tanh(t(a1))^2)
    w1 <<- w1 - l * t(d1) %*% t(c(1, input))
  }
}

save(w1, w2, logsumexp, softmax, neuro_func, file="nnet_image.rdata")
```