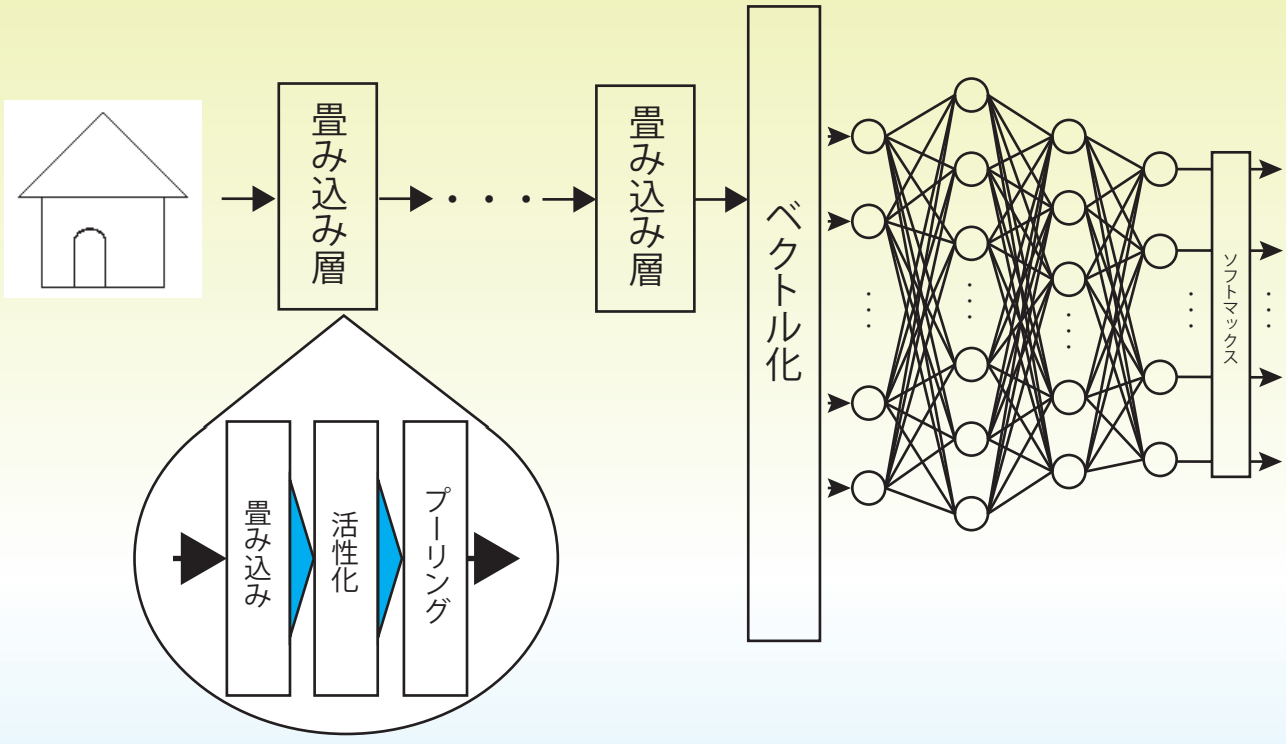
数学者が書いた深層学習講義

畳み込みニューラルネット篇

MATLAB プログラム付



新井 仁之

発行　数学と錯視の科学館 電子出版部

数学者が書いた深層学習講義

畳み込みニューラルネット篇

Ver. 1.5 （Appendix Ver. 1.3）

新井仁之
早稲田大学

概要　大学 1 年で学ぶ微分と行列を使って，特に畳み込みニューラルネットワークの深層学習の基礎事項を数学的厳密性を重視して解説する．数学はどのようになっているのだろうか，という疑問を持たれた場合に参考になることを目指している．**適宜，改訂版・増補版を発信していく予定．更新は www.araiweb.matrix.jp/DeepLearning.html に掲載する．**
MATLAB によるプログラムも上記サイトにあるので参考になれば幸いである．

1　まえがき

畳み込みニューラルネットによる深層学習が，画像分類ですばらしい能力を発揮していることは，今では良く知られている．ここでは深層学習の数学を丁寧に解説していこうと思う．なお，本稿は教育的なレクチャーノートであることから，畳み込みニューラルネットの深層学習の歴史については他書に譲ることにしたい．

まず，基本的な仕組みを概観しておく．畳み込みニューラルネットは，人（あるいはサル，ネコなどの動物）の脳内の視覚情報処理の仕組みの一部と多層パーセプトロンを組み合わせた写像である．写像の入力は画像で，出力はその画像が予め設定したクラスの何れに分類されるかを示す確率である．

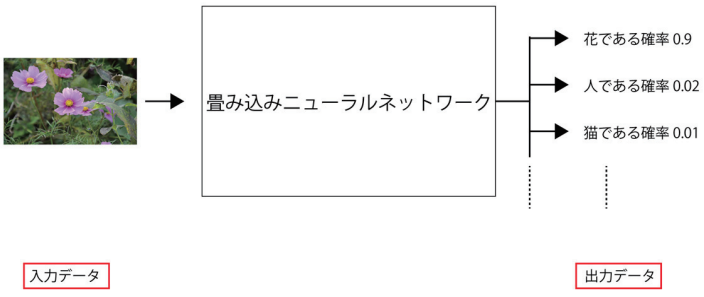


図 1　畳み込みニューラルネットは写像の一つ．

この写像の仕組みを解説するのが本稿の目的である．

2 数学からの準備

2.1 記号

今後しばしば使う記号を導入しておく．

$\mathbb{N} = \{1, 2, 3, \cdots\}$，$\mathbb{Z} = \{$ 整数全体 $\}$，$\mathbb{R} = \{$ 実数全体 $\}$ とし，$N \in \mathbb{N}$ に対して，$\mathbb{Z}_N = \{1, \cdots, N\}$ とする．

二つの集合 E, F に対して，その直積集合を

$$E \times F = \{(x, y) : x \in E, y \in F\}$$

により表す．

$\mathbb{R}^{M \times N}$ で M 行 N 列の実数の配列（行列）全体のなす集合を表す（数学ではしばしば $\boldsymbol{M}(M, N; \mathbb{R})$ 等と書かれる）．これを本稿では $M \times N$ **配列**と呼ぶ．$\mathbb{R}^N = \mathbb{R}^{N \times 1}$ で表す．M 行 N 列の配列 A に対して A^T で A の転置を表す．配列 A は各成分を $A(i, j)$ で表すならば，次のような配列である..

$$A = \begin{pmatrix} A(1,1) & \cdots & A(1,N) \\ \vdots & \ddots & \vdots \\ A(M,1) & \cdots & A(M,N) \end{pmatrix}$$

また

$$A^T = \begin{pmatrix} A(1,1) & \cdots & A(M,1) \\ \vdots & \ddots & \vdots \\ A(1,N) & \cdots & A(M,N) \end{pmatrix}$$

である．記載スペースを節約するために

$$A = (A(i,j)),$$
$$A = (A(i,j) : i \in \mathbb{Z}_M,\ j \in \mathbb{Z}_N)$$

のように略記する．

また，同様にして，集合 $E_1, \cdots, E_k$ に対して，その直積集合

$$E_1 \times \cdots \times E_k = \left\{ (x_1, \cdots, x_k) : \begin{array}{c} x_i \in E_i, \\ i = 1, \cdots .k \end{array} \right\}$$

も考える．

$N_1, \cdots, N_k \in \mathbb{N}$ とする．実数 $A(n_1, \cdots, n_k)$ の配列

$$A = (A(n_1, \cdots, n_k) : (n_1, \cdots, n_k) \in \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_k})$$

を $N_1 \times \cdots \times N_k$ **配列**という．$N_1 \times \cdots \times N_k$ 配列全体のなす集合を

$$\mathbb{R}^{N_1 \times \cdots \times N_k}$$

により表す．ニューラルネットの話では，この多次元の配列が頻繁に使われる．

2.2 畳み込みまたは相関積

畳み込みの意味は直ぐ後で説明することにして，まずは畳み込みの定義をしておこう．

$F=(F(p,q))\in\mathbb{R}^{m_1\times m_2}$, $X=(X(p,q))\in\mathbb{R}^{n_1\times n_2}$ とする．ただし $m_1\le n_1$，$m_2\le n_2$ とする．このとき

$$(F\bigstar X)(j,k)=\sum_{\mu=1}^{m_1}\sum_{\nu=1}^{m_2}F(\mu,\nu)X(j+\mu-1,k+\nu-1)$$

を F と X の**相関積**という．なお深層学習の分野では，これを**畳み込み**ということが多いようである．**本稿でも相関積のことを畳み込みということにする．**

ここで，$j+\mu-1$ と $k+\nu-1$ は，j,μ,k,ν によっては X の定義域をはみ出ることがあるので，j,k としては

$$1\le j\le n_1-m_1+1,\ 1\le k\le n_2-m_2+1$$

とする．こうしておけば

$$1\le j+\mu-1\le n_1-m_1+1+m_1-1=n_1$$
$$1\le k+\nu-1\le n_2-m_2+1+m_2-1=n_2$$

となっていて，この畳み込み積は well-defined である．

したがって

$$F\bigstar X\in\mathbb{R}^{(n_1-m_1+1)\times(n_2-m_2+1)}$$

となっている．なおこの畳み込みは **valid 型**と呼ばれている．他にも same 型というのがあり，それについては後述する．

■畳み込みの計算について　画像処理的には，F がフィルタで，X が画像データになる．$(j,k)=(1,1)$ の場合は

$$(F\bigstar X)(1,1)=\sum_{\mu=1}^{m_1}\sum_{\nu=1}^{m_2}F(\mu,\nu)X(\mu,\nu),$$

$(j,k)=(1,2)$ の場合は

$$(F\bigstar X)(1,2)=\sum_{\mu=1}^{m_1}\sum_{\nu=1}^{m_2}F(\mu,\nu)X(\mu,\nu+1)$$

$(j,k)=(1,3)$ の場合は

$$(F\bigstar X)(1,3)=\sum_{\mu=1}^{m_1}\sum_{\nu=1}^{m_2}F(\mu,\nu)X(\mu,\nu+2)$$

などとなっている．簡単な例を図で示せば，畳み込みは図 2 のように，フィルタで画像を縦，横に 1 コマずらしながらスキャンしている計算になっている．

図 2 の例の場合の計算を見ておこう．まず一つ目のブロック（赤点線で囲った部分）とフィルタの各成分の積の和，つまり内積を計算する：

$$\begin{bmatrix}0&1&0\\1&1&1\\0&1&0\end{bmatrix}\bullet\begin{bmatrix}1&2&3\\7&8&9\\13&14&15\end{bmatrix}$$
$$=0\times1+1\times2+0\times3+1\times7+1\times8+1\times9+0\times13+1\times14+0\times15$$
$$=40.$$

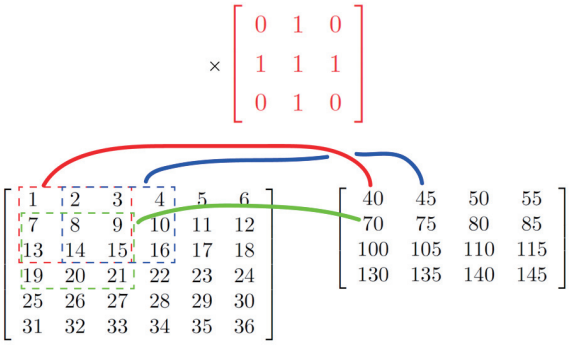


図 2　畳み込みの計算.

次に二つ目のブロック（青点線で囲った部分）とフィルタの内積をとる：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} 2 & 3 & 4 \\ 8 & 9 & 10 \\ 14 & 15 & 16 \end{bmatrix}$$
$$= 0 \times 2 + 1 \times 3 + 0 \times 4 + 1 \times 8 + 1 \times 9 + 1 \times 10 + 0 \times 14 + 1 \times 15 + 0 \times 16$$
$$= 45.$$

この計算を続けて，横の最後に行ったら，今度は一つ下にずらし，五つ目のブロック（緑点線で囲った部分）とフィルタとの内積をする：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} 7 & 8 & 9 \\ 13 & 14 & 15 \\ 19 & 20 & 21 \end{bmatrix} = 70.$$

この計算を続行していくと，畳み込みの結果は次のようになる

$$\begin{bmatrix} 40 & 45 & 50 & 55 \\ 70 & 75 & 80 & 85 \\ 100 & 105 & 110 & 115 \\ 130 & 135 & 140 & 145 \end{bmatrix}$$

■畳み込みの画像処理及び視覚情報処理的な意味　畳み込みの演算は画像処理や数理的な視覚の研究でもよく使われてきたものである．この演算により画像のさまざまな特徴を抽出できることが知られている．その一例をここで示しておこう．

いま図 3 のような画像を例にとって説明する．

これは 128×128 画素のグレースケール画像で，これを X とおく．$X \in \mathbb{R}^{128\times128}$ である．これと次の配列（フィルタ）

$$F = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{3\times3}$$

との畳み込み $F\bigstar W \in \mathbb{R}^{126\times126}$ を計算する．この画像を表示すると図 4 のようになっている．X のうち，**縦方向の成分が抽出できている**ことがわかるであろう．（斜め成分は縦方向の成分も含んでいるので，残っている.）

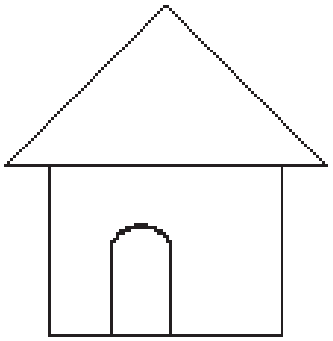


図 3　128 × 128 画素のグレースケール画像.

図 4　F との畳み込み後の画像．ただし見やすくするためにスケーリングして表示している．

もう一つ例を示しておく．今度は

$$F^T = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

と X との畳み込み $F^T \bigstar X \in \mathbb{R}^{126\times 126}$ である．結果は図 5 のようになる．図 4 に比べ**横成分が抽出されている**ことが確認できる．

F，F^T は**ソーベル・フィルタ**と呼ばれているもので，画像から一定の方向の成分を抽出する機能をもっている．他にも画像の特徴を抽出するフィルタは数多く知られている．

このように，特定のフィルタと呼ばれる配列との畳み込みにより，画像のある種の特徴を抽出できる．

じつは，ヒューベルとウィーセルという生理学者により，1959 年頃にネコ（後にマカクザル）の大脳皮質 V1 野に，ソーベル・フィルタのようなプロファイルを受容野とする細胞が存在することが確認された．この細胞は**単純細胞**と名付けられた．このことから，脳のこの部分の数理モデルとして畳み込みが用いられるようになった．なおその後の視覚科学の研究により，フィルタとしてはソーベル・フィルタよりもより複雑なフィルタが数理モデルとしては適切であることがわかってきた．この辺の話については，稿を改めて議論したい．

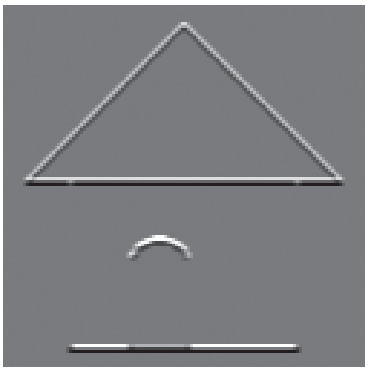


図 5　F^T との畳み込み後の画像．ただし見やすくするためにスケーリングして表示している．

練習 1 $X = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$ とし，$F = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$ とする．このとき，$F\bigstar X$ を計算せよ．

解答 $F\bigstar X = \begin{bmatrix} 27 & 38 \\ 34 & 21 \end{bmatrix}$.

2.3　ゼロ・パディングによる畳み込み - same 型畳み込み

前節の畳み込みでは，畳み込みをするとサイズが小さくなる．同じサイズに保つ方法として画像サイズをゼロ・パディングで拡張して畳み込みを行う方法もある．

$F = (F(u,v)) \in \mathbb{R}^{m_1 \times m_2}$ とする．$p_i = \mathrm{fix}(m_i/2)$ とする（fix は与えられた小数を 0 に近い整数に丸めこむ操作を表している）．$X = (X(p,q)) \in \mathbb{R}^{n_1 \times n_2}$ とする．$\widetilde{X} \in \mathbb{R}^{(n_1+2p_1)\times(n_2+2p_2)}$ を

$$\widetilde{X}(\alpha,\beta) = \begin{cases} X(\alpha - p_1, \beta - p_2), & (\alpha,\beta) \in [p_1+1, p_1+n_1] \times [p_2+1, p_2+n_2] \\ 0, & \text{その他} \end{cases}$$

と定める（ここで，$[p_1+1, p_1+n_1] \times [p_2+1, p_2+n_2]$ は閉区間の直積です）．たとえば $m_1 = m_2 = 3$, $n_1 = n_2 = 4$ とする．$p_1 = p_2 = 1$ である．このとき，$[p_1+1, p_1+n_1] \times [p_2+1, p_2+n_2] = [2,5] \times [2,5]$ となる．$(\alpha,\beta) \in [2,5] \times [2,5]$ のとき，$\widetilde{X}(\alpha,\beta) = X(\alpha-1, \beta-1)$ であるから，

$$\widetilde{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & X(1,1) & X(1,2) & X(1,3) & X(1,4) & 0 \\ 0 & X(2,1) & X(2,2) & X(2,3) & X(2,4) & 0 \\ 0 & X(3,1) & X(3,2) & X(3,3) & X(3,4) & 0 \\ 0 & X(4,1) & X(4,2) & X(4,3) & X(4,4) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

である．

そして F と $\widetilde{X}$ との畳み込み $F\bigstar X$ を考える．$\widetilde{X} \in \mathbb{R}^{(n_1+2p_1-m_1+1)\times(n_2+2p_2-m_2+1)}$ となっている．ここで次の二つの場合を考える．

■m_i **が奇数の場合．**　$m_i = 2k+1$ とおくと，$p_i = \mathrm{fix}(k+1/2) = k$ である．したがって，$n_1+2p_1-m_1+1 = n_1$.

■m_i **が偶数の場合．**　$m_i = 2k$ とおくと，$p_i = \mathrm{fix}(k) = k$ である．したがって，$n_1+2p_1-m_1+1 = n_1+1$.

偶数の場合はサイズが $n_1 \times n_2$ よりも 1 大きくなってしまう．そのため畳み込みの結果が X のサイズと同じになるように，$\widetilde{X}$ を $[1, n_1] \times [1, n_2]$ の範囲に制限する．

以上を

$$F\bigstar_{\mathrm{zp}}X$$

と表し，**same 型の畳み込み**と呼ぶ．具体的には，$1 \le j \le n_1,, 1 \le k \le n_2$ に対して

$$(F\bigstar_{\mathrm{zp}}X)(j,k) = \sum_{\mu=1}^{m_1}\sum_{\nu=1}^{m_2} W(\mu,\nu)\widetilde{X}(j+\mu-1, k+\nu-1)$$

である．

例を計算してみよう．

練習 2 $X = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$ とし，$F = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$ とする．このとき，$F\bigstar_{\mathrm{zp}}X$ を計算せよ．また $G = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ の場合に $G\bigstar_{\mathrm{zp}}X$ を計算せよ．

解答

$$\widetilde{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 16 & 2 & 3 & 13 & 0 \\ 0 & 5 & 11 & 10 & 8 & 0 \\ 0 & 9 & 7 & 6 & 12 & 0 \\ 0 & 4 & 14 & 15 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

であり，

$$F\bigstar_{\mathrm{zp}}X = \begin{bmatrix} 24 & 20 & 16 & 8 \\ 30 & 27 & 38 & 19 \\ 31 & 34 & 21 & 25 \\ 27 & 40 & 36 & 28 \end{bmatrix}.$$

$$G\bigstar_{\mathrm{zp}}X = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$$

3　畳み込みニューラルネットワーク

3.1　概略

■**畳み込みニューラルネットの骨組み**　畳み込みニューラルネットワークは，まず入力画像をいくつかのフィルタにより，**フィルタリング（畳み込み）**する．それから活性化関数で**活性化**する．これは神経細胞の発火の

仕組みをモデル化して発展させたものである．本稿では後述の ReLU 関数と呼ばれるものを使う．それから後述する**プーリング**をする．プーリングとはある種のダウンサンプリングである．この一連の操作をするのが畳み込み層と呼ばれる．このたたみ込み層を必要に応じていくつか直列する．その後，画像データをベクトルに配列し直して，次に**全結合層**を置く．全結合層はすでに [1] で解説したものである．

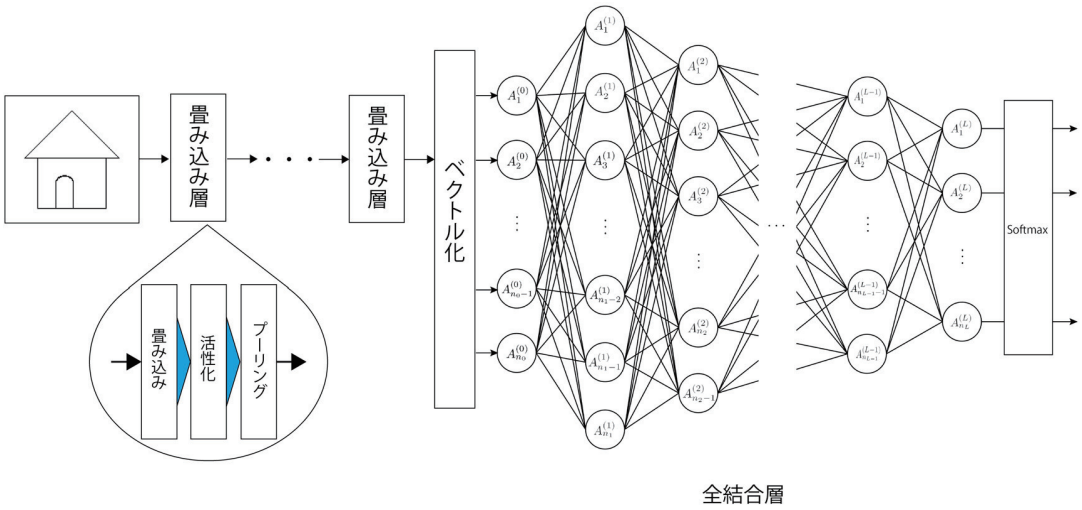


図 6　畳み込みニューラルネットの骨組みの例

■誤差によるフィルタと重み，バイアスの修正　本稿ではいわゆる**教師あり学習**を扱う．教師あり学習には，学習のためのデータが正解付きで用意されている（あるいは自ら用意する）．これを訓練データとテストデータに分ける（あるいはさらに検証用データも準備するが，本稿では扱わない）．

最初に畳み込み層のフィルタとバイアス，全結合層のフィルタとバイアスを定める．そして訓練データを入力する．その結果と正解データを比較する．通常は出力結果と正解データとの間には誤差がある．次に，その誤差を小さくするように，各層のフィルタ/重み，バイアスを変更する．変更にあたっては，たとえば後述の勾配降下法を使う．またそのための勾配のは後述の誤差逆伝播法を用いる．

そしてこの操作を何度か繰り返し，誤差が十分小さくなるようにする．

これで学習が終わる．次に学習が適切に行われたかを調べるためにテストデータを使って，今度は正解率を見る．（正解率が悪いようであれば，適切に修正して，以上の工程を繰り返す．）

これが畳み込みニューラルネットの教師あり深層学習の例の概略である．**以下ではこれを数学的に厳密に解説していきたい．**

3.2　順伝播

畳み込み層が2 層あり，全結合層が 2 層ある場合を考える．ここを一般的にしてもよいが，かえってわかりにくくなると思うので，一般的な場合の本質を含む上記の具体的な場合について解説することにしたい．畳み込み層が L_1 層あり，全結合層が L_2 層にすることは，形式的な一般化なので容易である．

■第 0 層　入力データを

$$x^{(0)} = (x^{(0)}(c,i,j)) \in \mathbb{R}^{c_0 \times h_0 \times w_0}$$

とする．ここで c はチャネル（**チャネル数** c_0），i,j が画素の位置座標（**画像サイズ** $h_0 \times w_0$）である．チャネル数は，たとえばグレースケールの画像ならば 1，カラー画像ならば通常 3 （たとえば R，G，B に対応）である．以上のパラメータをまとめておくと

$$c = 1, \cdots, c_0, \ \begin{cases} i = 1, \cdots, h_0 \\ j = 1, \cdots, w_0 \end{cases}$$

である．

■第 1 層　ここでは畳み込み処理を行う．畳み込みのための**フィルタ**を

$$F^{(1)} = \left(F^{(1)}(k,c,\mu,\nu)\right) \in \mathbb{R}^{c_1 \times c_0 \times m^{(1)} \times m^{(1)}}$$

とする．ここで，k はフィルタ番号（フィルタの数は c_1），c は入力画像のチャネル（チャネル数は，第 0 層の画像数 c_0），μ,ν はフィルタの値の位置座標（フィルタサイズは $m^{(1)} \times m^{(1)}$）である．すなわち，

$$k = 1, \cdots, c_1, \ c = 1, \cdots, c_0, \ \begin{cases} \mu = 1, \cdots, m^{(1)} \\ \nu = 1, \cdots, m^{(1)} \end{cases}$$

とする．ここでフィルタは正方配列とするが，これは長方形配列としても同様にできる．また，**バイアス**を

$$b^{(1)}_{\mathrm{conv}} = (b^{(1)}_{\mathrm{conv}}(k)) \in \mathbb{R}^{c_1}$$

（すなわち $k = 1, \cdots, c_1$）とする．

次の順で入力層からのデータの処理をする．以下，

$$f(x) = \max\{0, x\} \quad (x \in \mathbb{R})$$

とする（これは **ReLU 関数**と呼ばれている）．$f(x)$ は活性化関数と呼ばれるものの一つである．活性化関数は神経科学に由来を持つが，それについては [3, 12.4 節] で解説してあるので参照してほしい．まとめると次のように操作が進む：

$$\boxed{x^{(0)} \text{ が入力される}}$$
$$\Downarrow$$
$$\boxed{u^{(1)}_{\mathrm{conv}} = F^{(1)} \bigstar x^{(0)} + b^{(1)}_{\mathrm{conv}};\ \text{畳み込み}}$$
$$\Downarrow$$
$$\boxed{a^{(1)}_{\mathrm{conv}} = f(u^{(1)}_{\mathrm{conv}});\ \text{活性化}}$$
$$\Downarrow$$
$$\boxed{p^{(1)}_{\mathrm{conv}};\ \text{プーリング}}$$
$$\Downarrow$$
$$\boxed{x^{(1)}_{\mathrm{conv}} = p^{(1)}_{\mathrm{conv}};\ \text{出力}}$$

ここで，各操作によるデータサイズの変化も明記しておく．（なお畳み込みの種類としてはいくつかあるが，本稿では valid 型で説明する．）

入力画像データは $x^{(0)} = \left(x^{(0)}(c,i,j)\right) \in \mathbb{R}^{c_0 \times h_0 \times w_0}$ である．**畳み込み**（正確には畳み込み + バイアス）は

$$u^{(1)}_{\mathrm{conv}}(k,i,j) = \sum_{c=1}^{c_0} \sum_{\mu=1}^{m^{(1)}} \sum_{\nu=1}^{m^{(1)}} F^{(1)}(k,c,\mu,\nu) x^{(0)}(c, i+\mu-1, j+\nu-1) + b^{(1)}_{\mathrm{conv}}(k)$$

で定義され，$k = 1, \cdots, c_1$，$i = 1, \cdots, h_0 - m^{(1)} + 1$，$j = 1, \cdots, w_0 - m^{(1)} + 1$ である．ここで畳み込みによる**出力サイズ**を

$$h'_0 = h_0 - m^{(1)} + 1$$
$$w'_0 = w_0 - m^{(1)} + 1$$

とおくと，

$$u^{(1)}_{\mathrm{conv}} = (u^{(1)}_{\mathrm{conv}}(k,i,j)) \in \mathbb{R}^{c_1 \times h'_0 \times w'_0}$$

である．畳み込みしたものを**活性化**する．

$$a^{(1)}_{\mathrm{conv}} = (a^{(1)}_{\mathrm{conv}}(k,i,j)) = (f(u^{(1)}_{\mathrm{conv}}(k,i,j))) \in \mathbb{R}^{c_1 \times h'_0 \times w'_0}$$

である．サイズは $u^{(1)}_{\mathrm{conv}}$ と同じである．

プーリングは次のように行われる．これは下記の四つの値 $a^{(1)}(\cdot,\cdot)$ のうちの最大のものをとるというものである．

$$p^{(1)}_{\mathrm{conv}}(k,i',j') = \max\Big\{a^{(1)}(k,2i'-1,2j'-1), a^{(1)}(k,2i'-1,2j'),$$
$$a^{(1)}(k,2i',2j'-1), a^{(1)}(k,2i',2j')\Big\}.$$

ここで $k = 1, \cdots, c_1$，$i' = 1, \cdots, h_1$，$j' = 1, \cdots, w_1$，ただし，

$$h_1 = \mathrm{fix}(h'_0/2),$$
$$w_1 = \mathrm{fix}(w'_0/2)$$

である．したがって

$$p^{(1)}_{\mathrm{conv}} = (p^{(1)}(k,i',j')) \in \mathbb{R}^{c_1 \times h_1 \times w_1}$$

である．なおこのプーリングは**最大プーリング**と呼ばれるもので，このほかにもプーリングの方法はいくつか知られている．その一つは**平均プーリング**で，これは

$$p^{(1)}_{\mathrm{ave}}(k,i',j') = \frac{1}{4}\Big(a^{(1)}(k,2i'-1,2j'-1) + a^{(1)}(k,2i'-1,2j')$$
$$a^{(1)}(k,2i',2j'-1), a^{(1)}(k,2i',2j')\Big)$$

により定義される．

第1 層の出力サイズは $h_1 \times w_1$ である．これに至る第 1 層でのデータサイズの変化を記しておく：

$$\underset{\text{入力}}{h_0 \times w_0} \longrightarrow \underset{\text{畳み込み出力}}{h'_0 \times w'_0} \longrightarrow \underset{\text{畳み込み層出力}}{h_1 \times w_1}$$

【最大プーリングで残した座標番号の記号の設定】

最大プーリングの定義から，$p^{(1)}_{\mathrm{conv}}(k,i',j') = a^{(1)}_{\mathrm{conv}}(k,\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')})$ をみたすような

$$\left(\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')}\right) \in \{(2i'-1,2j'-1),(2i'-1,2j'),(2i',2j'-1),(2i',2j')\}$$

が少なくとも一つ存在する．その一つ定め固定する．ここで

$$(i',j') \in [1,h_1]\times[1,w_1]$$

であり，

$$\left(\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')}\right) \in [1,h'_0]\times[1,w'_0]$$

である（図 7 参照）．

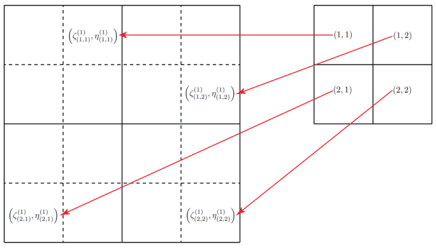


図 7　最大プーリングの位置

ここで，$\zeta^{(1)}_{(i',j')} = i$，$\eta^{(1)}_{(i',j')} = j$ とおくとき，$\alpha^{(1)}_{(i,j)} = i'$，$\beta^{(1)}_{(i,j)} = j'$ と定める．図 7 で言えば，

$$\left(\zeta^{(1)}_{(1,1)},\eta^{(1)}_{(1,1)}\right) = (1,2), \quad \left(\zeta^{(1)}_{(1,2)},\eta^{(1)}_{(1,2)}\right) = (2,4),$$
$$\left(\zeta^{(1)}_{(2,1)},\eta^{(1)}_{(2,1)}\right) = (4,1), \quad \left(\zeta^{(1)}_{(2,2)},\eta^{(1)}_{(2,2)}\right) = (4,4)$$

であるから，

$$\left(\alpha^{(1)}_{(1,2)},\beta^{(1)}_{(1,2)}\right) = (1,1), \quad \left(\alpha^{(1)}_{(2,4)},\beta^{(1)}_{(2,4)}\right) = (1,2),$$
$$\left(\alpha^{(1)}_{(4,1)},\beta^{(1)}_{(4,1)}\right) = (2,1), \quad \left(\alpha^{(1)}_{(4,4)},\beta^{(1)}_{(4,4)}\right) = (2,2),$$

である．

最後に出力データは

$$x^{(1)}_{\mathrm{conv}} = \left(x^{(1)}_{\mathrm{conv}}(k,i',j')\right) = \left(p^{(1)}_{\mathrm{conv}}(k,i',j')\right) \in \mathbb{R}^{c_1\times h_1\times w_1}$$

である．パラメータを書き直して

$$x^{(1)}_{\mathrm{conv}} = \left(x^{(1)}_{\mathrm{conv}}(c,i,j)\right) \in \mathbb{R}^{c_1\times h_1\times w_1}$$

としておく．ただし，$c = 1,\cdots,c_1$，$i = 1,\cdots,h_1$，$j = 1,\cdots,w_1$ である．第 1 層ではフィルタ番号だったものが，出力では出力画像データのチャネルとなる．

■**第 2 層**　第 2 層には，第 1 層からの出力 $x^{(1)}_{\rm conv}=\left(x^{(1)}_{v\rm conv}(c,i,j)\right)\in\mathbb{R}^{c_1\times h_1\times w_1}$ が入力データとなる．

第 2 層でも畳み込み処理を行う．畳み込みのフィルタとバイアスは次のようなものとする．**フィルタ**は

$$F^{(2)}=\left(F^{(2)}(k,c,\mu,\nu)\right)\in\mathbb{R}^{c_2\times c_1\times m^{(2)}\times m^{(2)}}$$

とするが，ここで，k はフィルタ番号（フィルタ数は c_2），c は入力データのチャネル（チャネル数は c_1），μ,ν はフィルタの数値の位置座標である（フィルタサイズは $m^{(2)}\times m^{(2)}$）．

バイアスは

$$b^{(2)}_{\rm conv}=\left(b^{(2)}_{\rm conv}(k)\right)\in\mathbb{R}^{c_2}$$

である．

第 2 層の**畳み込み**は次のように定義される．

$$u^{(2)}_{\rm conv}(k,i,j)=\sum_{c=1}^{c_1}\sum_{\mu=1}^{m^{(2)}}\sum_{\nu=1}^{m^{(2)}}F^{(2)}(k,c,\mu,\nu)x^{(1)}_{\rm conv}(c,i+\mu-1,j+\nu-1)+b^{(2)}_{\rm conv}(k)$$

ただしここで，$k=1,\cdots,c_2$，$i=1,\cdots,h_1-m^{(2)}+1$，$j=1,\cdots,w_1-m^{(2)}+1$ である．以下，**出力サイズ**を

$$h'_1=h_1-m^{(2)}+1$$
$$w'_1=w_1-m^{(2)}+1$$

とおく．したがって

$$u^{(2)}_{\rm conv}=\left(u^{(2)}_{\rm conv}(k,i,j)\right)\in\mathbb{R}^{c_2\times h'_1\times w'_1}.$$

これを ReLU 関数で**活性化**する．

$$a^{(2)}_{\rm conv}=\left(a^{(2)}_{\rm conv}(k,i,j)\right)=\left(f(u^{(2)}_{\rm conv}(k,i,j))\right)\in\mathbb{R}^{c_2\times h'_1\times w'_1},$$

ここでデータサイズは $u^{(2)}_{\rm conv}$ と同じで，$k=1,\cdots,c_2$，$i=1,\cdots,h'_1$，$j=1,\cdots,w'_1$ である．

第 2 層の**プーリング**は次のように行われる．

$$p^{(2)}_{\rm conv}(k,i',j')=\max\Big\{a^{(2)}_{\rm conv}(k,2i'-1,2j'-1),a^{(2)}_{\rm conv}(k,2i'-1,2j'),$$
$$a^{(2)}_{\rm conv}(k,2i',2j'-1),a^{(2)}_{\rm conv}(k,2i',2j')\Big\}.$$

ただし $k=1,\cdots,c_2$，$i'=1,\cdots,h_2$，$j'=1,\cdots,w_2$，であり，ここで

$$h_2=\mathrm{fix}(h'_1/2),$$
$$w_2=\mathrm{fix}(w'_1/2)$$

としている．

最後に出力データは

$$x^{(2)}_{\rm conv}=\left(x^{(2)}_{\rm conv}(k,i',j')\right)=\left(p^{(2)}_{\rm conv}(k,i',j')\right)\in\mathbb{R}^{c_2\times h_2\times w_2}$$

である．パラメータを置き換えて

$$x^{(2)}_{\rm conv}=\left(x^{(2)}_{\rm conv}(c,i,j)\right)\in\mathbb{R}^{c_2\times h_2\times w_2}$$

と標記しておく．ここで，$c=1,\cdots,c_2$，$i=1,\cdots,h_2$，$j=1,\cdots,w_2$ である．第 1 層でフィルタ数だったものが，出力ではデータのチャネルとなる

さて，第 1 層と同様に $p^{(2)}_{\mathrm{conv}}(k,i',j')=a^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i',j')},\eta^{(2)}_{(i',j')})$ をみたすような

$$\left(\zeta^{(2)}_{(i',j')},\eta^{(2)}_{(i',j')}\right)\in\{(2i'-1,2j'-1),(2i'-1,2j'),(2i',2j'-1),(2i',2j')\}$$

が少なくとも一つ存在するので，その一つ固定しておく．ここで

$$(i',j')\in[1,h_2]\times[1,w_2]$$

であるが，

$$\left(\zeta^{(2)}_{(i',j')},\eta^{(2)}_{(i',j')}\right)\in[1,h_1']\times[1,w_1']$$

である．

以上をまとめると，**第2 層の出力サイズ**は $h_2\times w_2$ である．つまり第 2 層でのデータサイズの変化は

$$\underset{\text{入力}}{h_1\times w_1}\longrightarrow\underset{\text{畳み込み出力}}{h_1'\times w_1'}\longrightarrow\underset{\text{畳み込み層出力}}{h_2\times w_2}$$

である．

以上で畳み込み層の設定が終了である．
次に全結合層に移る．

■ベクトル化　第 2 層からの出力は $x^{(2)}_{\mathrm{conv}}=\left(x^{(2)}_{\mathrm{conv}}(c,i,j)\right)\in\mathbb{R}^{c_2\times h_2\times w_2}$ であるが，これを次のように 1 行に並べてベクトル化する：

$$\begin{aligned}v_{\mathrm{full}}=(&x^{(2)}_{\mathrm{conv}}(1,1,1),\cdots,x^{(2)}_{\mathrm{conv}}(1,1,w_2),\\&x^{(2)}_{\mathrm{conv}}(1,2,1),\cdots,x^{(2)}_{\mathrm{conv}}(1,2,w_2),\\&\cdots\cdots\\&x^{(2)}_{\mathrm{conv}}(1,h_2,1),\cdots,x^{(2)}_{\mathrm{conv}}(1,h_2,w_2),\\&\cdots\cdots\\&x^{(2)}_{\mathrm{conv}}(c_2,h_2,1),\cdots,x^{(2)}_{\mathrm{conv}}(c_2,h_2,w_2))^T.\end{aligned}$$

ベクトルの成分の総数を

$$n_2=c_2h_2w_2$$

とおけば，$v_{\mathrm{full}}\in\mathbb{R}^{n_2}=\mathbb{R}^{c_2h_2w_2}$ である．
このベクトル化したデータが第 2 層から出力される．

■第 3 層（全結合層の第 1 層）　第 3 層，すなわち全結合層の一層目である．入力信号は最後の畳み込み層からの出力データをベクトル化した $v_{\text{full}} \in \mathbb{R}^{n_2} = \mathbb{R}^{c_2 h_2 w_2}$ である．

第 3 層の重みを

$$W^{(1)} = (W^{(1)}(k,i)) \in \mathbb{R}^{c_3 \times n_2}$$

とする．ここで $k = 1, \cdots, c_3$ は重みの番号である．バイアスは

$$b^{(1)}_{\text{full}} = \left(b^{(1)}_{\text{full}}(i)\right) \in \mathbb{R}^{c_3}.$$

まず次のアフィン変換を施す：

$$u^{(1)}_{\text{full}}(k) = \sum_{i=1}^{n_1} W^{(1)}(k,i) v(i) + b^{(1)}_{\text{full}}(k)$$

$(k = 1, \cdots, c_3)$．行列表示を用いるならば

$$u^{(1)}_{\text{full}} = W^{(1)} v + b^{(1)}_{\text{full}} \in \mathbb{R}^{c_3}.$$

活性化は

$$a^{(1)}_{\text{full}} = \left(a^{(1)}_{\text{full}}(k)\right) = \left(f(u^{(1)}_{\text{full}}(k))\right) \in \mathbb{R}^{c_3}$$

である．ここで $k = 1, \cdots, c_3$ は第 3 層の重みの数であるが，これが出力データの長さになる．出力データを改めて

$$v^{(1)}_{\text{full}} = \left(v^{(1)}_{\text{full}}(k)\right) = \left(a^{(1)}_{\text{full}}(k)\right) \in \mathbb{R}^{c_3}$$

と記す．

■第 4 層（全結合層の第 1 層，出力層）　第 3 層からの入力は $v^{(1)}_{\text{full}} = \left(v^{(1)}_{\text{full}}(i)\right) \in \mathbb{R}^{c_3}$ である．第 4 層の重みを

$$W^{(2)} = \left(W^{(2)}(k,i)\right) \in \mathbb{R}^{c_4 \times c_3}$$

とする．ここで $k = 1, \cdots, c_4$ はフィルタ数であり，$i = 1, \cdots, c_3$．バイアスは

$$b^{(2)}_{\text{full}} = (b^{(2)}_{\text{full}}(k)) \in \mathbb{R}^{c_4}.$$

アフィン変換は

$$u^{(2)}_{\text{full}}(k) = \sum_{i=1}^{c_3} W^{(4)}(k,i) v^{(1)}_{\text{full}}(i) + b^{(2)}_{\text{full}}(k)$$

$(k = 1. \cdots, c_4)$ で定義される．

$$u^{(2)}_{\text{full}} = \left(u^{(2)}_{\text{full}}(k)\right) \in \mathbb{R}^{c_4}.$$

と表す．最後に分類のための**ソフトマックス関数**を施す．ソフトマックス関数は

$$\text{Soft}(u^{(2)}_{\text{full}})(k) = \frac{\exp(u^{(2)}_{\text{full}}(k))}{\sum_{j=1}^{c_4} \exp(u^{(2)}_{\text{full}}(j))}$$

により定義される．

$$0 \leq \mathrm{Soft}(u^{(2)}_{\mathrm{full}})(k) \leq 1$$
$$\sum_{k=1}^{c_4} \mathrm{Soft}(u^{(2)}_{\mathrm{full}})(k) = 1$$

なので，$\mathrm{Soft}(u^{(4)})(k)$ が入力画像が第 k クラスである確率とみなすことができる．

$$v^{(2)}_{\mathrm{out}} = \left(v^{(2)}_{\mathrm{out}}(k)\right) = \left(\mathrm{Soft}(u^{(2)}_{\mathrm{full}})(k)\right) \in \mathbb{R}^{c_4}$$

と定義する．

注意　ソフトマックス関数の値は，定数を加算しても変化しない．実際，$\alpha \in \mathbb{R}$ としたとき，$\mathbf{1} = (1, \cdots, 1)^T$ として

$$\begin{aligned}\mathrm{Soft}(u^{(2)}_{\mathrm{full}} + \alpha\mathbf{1})(k) &= \frac{\exp(u^{(2)}_{\mathrm{full}}(k) + \alpha)}{\sum_{j=1}^{c_4} \exp(u^{(2)}_{\mathrm{full}}(j) + \alpha)} = \frac{e^{\alpha}\exp(u^{(2)}_{\mathrm{full}}(k))}{e^{\alpha}\sum_{j=1}^{c_4} \exp(u^{(2)}_{\mathrm{full}}(j))} \\ &= \mathrm{Soft}(u^{(2)}_{\mathrm{full}})(k)\end{aligned}$$

である．e^x は x が大きくなると指数関数的に大きくなるため計算機の計算では不都合が生じる，これを避けるために，たとえばデータの最大値を引いて e^x の値が大きくなりすぎないようにすることが可能である．

■誤差測定　教師あり学習では，入力 $x^{(0)}$ に対する正解ベクトル $r = (r(1), \cdots, r(c_4))^T \in \mathbb{R}^{c_4}$ が与えられている．ここで $r(k)$ は，$x^{(0)}$ が第 k クラスに属する場合は 1，そうでない場合は 0 とする．

正解ベクトル r と $v^{(4)}$ との誤差を測る．通常，交差エントロピーで測定する．これは

$$E_{x^{(0)}} = -\sum_{k=1}^{c_4} r(k) \log\left(v^{(2)}_{\mathrm{out}}(k)\right)$$

で定義される．

すべての誤差を集めたものを全体の誤差と考える場合もあるが，それよりもよく使われるのが，**ミニバッチ法**と呼ばれる方法である．この方法では，学習データ全体の集合を，いくつかのミニバッチと呼ばれる部分集合に（たとえばランダムに）分割する．たとえば，1000 個ある学習データを 100 個からなる 10 個の部分集合に分割するといった具合である．

いま，学習データ全体のなす集合を $\mathcal{L}$ とし，ミニバッチを

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_1 \cup \cdots \cup \mathcal{L}_M \\ \mathcal{L}_i \cap \mathcal{L}_j &= \varnothing \ (i \neq j) \\ \mathcal{L}_i &\neq \varnothing \ (i = 1, \cdots, M)\end{aligned}$$

とする．各 $\mathcal{L}_i$ を構成する要素の数を $\#(\mathcal{L}_i)$ で表す．誤差関数として

$$E_i = \frac{1}{\#(\mathcal{L}_i)} \sum_{x \in \mathcal{L}_i} E_x$$

を考える（$i = 1, \cdots, M$）．

3.3 誤差を小さくする方法

畳み込みニューラルネットにおいて最も重要な問題は

フィルタ/重み，バイアスをどのようにとれば，正解データと出力データの誤差を小さくできるか？

ということである．誤差関数 E_i はフィルタ/バイアス，重みの関数と考えられるから，これらを変数とする多変数関数の極値問題を考えればよい．

考えている畳み込みニューラルネットのすべてのフィルタを集めた集合を $\mathcal{F}$，すべての重みを集めた集合を $\mathcal{W}$，すべてのバイアスを集めた集合を $\mathcal{B}$ と表す．今の場合，

$$\mathcal{F} = \{F^{(1)}, F^{(2)}\}$$
$$\mathcal{W} = \{W^{(1)}, W^{(2)}\}$$
$$B = \{b^{(1)}_{\text{conv}}, b^{(2)}_{\text{conv}}, b^{(1)}_{\text{full}}, b^{(2)}_{\text{full}}\}$$

である．これらを変数とする E_i の勾配を $\nabla_{\mathcal{F},\mathcal{W},\mathcal{B}} E_i$ と表す．多変数関数の極値問題の話から（たとえば [3] 参照），もしも $\nabla_{\mathcal{F},\mathcal{W},\mathcal{B}} E_i = \mathbf{0}$ となる $\mathcal{F}, \mathcal{W}, \mathcal{B}$ を見いだせれば，それが極値の候補である．

しかし，これを直接見つけるのは非常に難しい．E_i が複雑であるし，変数の数も多い．そこで深層学習では，次の二つの方法を組み合わせて実行する．

1．勾配降下法
2．誤差逆伝播法

■勾配降下法　勾配降下法の詳しい説明は，[3] にもあるが，詳細な解説は改めて別稿で行うことにするが，要するに，小さな数 $\varepsilon > 0$ を定めて，次を計算すればよい．ここで ε は**学習率**と呼ばれている．簡単のため $\mathcal{L}_1$ の誤差 E_1 を E で表す．そして次のようにパラメータ更新を行う．

【パラメータ更新】

$$F^{(1)}_{\text{new}} = F^{(1)} - \varepsilon \nabla_{F^{(1)}} E,$$
$$F^{(2)}_{\text{new}} = F^{(2)} - \varepsilon \nabla_{F^{(2)}} E,$$
$$W^{(1)}_{\text{new}} = W^{(1)} - \varepsilon \nabla_{W^{(1)}} E,$$
$$W^{(2)}_{\text{new}} = W^{(2)} - \varepsilon \nabla_{W^{(2)}} E,$$

$$b^{(i)}_{\text{conv,new}} = b^{(i)}_{\text{conv}} - \varepsilon \nabla_{b^{(i)}_{\text{conv}}} E \ \ (i = 1, 2)$$
$$b^{(i)}_{\text{full,new}} = b^{(i)}_{\text{full}} - \varepsilon \nabla_{b^{(i)}_{\text{full}}} E \ \ (i = 1, 2)$$

次に更新されたフィルタ/ウェイト，バイアスを改めて $F^{(1)}$, $F^{(2)}$, $W^{(1)}$, $W^{(2)}$, $b^{(i)}_{\text{conv}}$, $b^{(i)}_{\text{full}}$ $(i = 1, 2)$ と置きなおして，$\mathcal{L}_2$ の誤差 E_2 を計算し，$E = E_2$ として，【パラメータの更新】のプロセスを行う．以下，このプロセスを繰り返して，$\mathcal{L}_3, \cdots, \mathcal{L}_M$ まで繰り返す．ここまでの工程を**エポック**という．

必要なら（つまり誤差の減少が足りなければ）さらにエポックを追加して行う．

何回かエポックを繰り返しても，誤差の適切な減少が見られない場合の対処法の例としては，畳み込みニューラルネットの骨組みを検討したり（つまり層の数やフィルタ/重みの数を増やしたり，減らしたり），学

習率を変える．勾配降下法を改良した別の方法，たとえばモメント法等々を試すという手段などがある．さらに別の工夫が必要になることもある．

■誤差逆伝播法　【パラメータの更新】のプロセスで本質的なものは，それぞれの勾配ベクトルの計算である．勾配ベクトルは本質的に偏微分の計算であるが，偏微分の計算はコンピュータに載りにくい．そこで考えられたのが誤差逆伝播法である．これは出力層から逆順に計算して言うのであるが，偏微分の計算を巧妙に避け，代数的な計算で済むようにされている．以下では，誤差逆伝播法の数学を解説していく．

■第 4 層からの逆伝播　$k = 1, \cdots, c_4,\ i = 1, \cdots, c_3$ として，

$$\begin{aligned}\frac{\partial E}{\partial W^{(2)}(k,i)} &= \sum_{j=1}^{c_4} \frac{\partial E}{\partial u^{(2)}_{\text{full}}(j)} \frac{\partial u^{(2)}_{\text{full}}(j)}{\partial W^{(4)}(k,i)} \\ &= \sum_{j=1}^{c_4} \frac{\partial E}{\partial u^{(2)}_{\text{full}}(j)} \frac{\partial}{\partial W^{(2)}(k,i)} \left(\sum_{j'=1}^{c_3} W^{(2)}(j,j') v^{(1)}(j') + b^{(2)}_{\text{full}}(j) \right) \\ &= \frac{\partial E}{\partial u^{(1)}_{\text{full}}(k)} v^{(1)}(j).\end{aligned}$$

ここで

$$\delta^{(2)}_{\text{full}}(k) = \frac{\partial E}{\partial u^{(2)}_{\text{full}}(k)}$$

とおくと，

$$\frac{\partial E}{\partial W^{(2)}(k,i)} = \delta^{(2)}_{\text{full}}(k) v^{(1)}(j)$$

である．また，同様にして

$$\begin{aligned}\frac{\partial E}{\partial b^{(2)}_{\text{full}}(k)} &= \sum_{j=1}^{c_4} \frac{\partial E}{\partial u^{(2)}_{\text{full}}(j)} \frac{\partial}{\partial b^{(2)}_{\text{full}}(k)} \left(\sum_{j'=1}^{c_3} W^{(2)}(j,j') v^{(1)}(j') + b^{(2)}_{\text{full}}(j) \right) \\ &= \frac{\partial E}{\partial u^{(2)}_{\text{full}}(j)} = \delta^{(2)}_{\text{full}}(k).\end{aligned}$$

以上をまとめると

$$\begin{aligned}\nabla_{W^{(2)}} E &= \left(\frac{\partial E}{\partial W^{(2)}(k,i)} \right) = \left(\delta^{(2)}_{\text{full}}(k) v^{(1)}(j) \right) = \delta^{(2)}_{\text{full}} v^{(1)T} \in \mathbb{R}^{c_4 \times c_3} \\ \nabla_{b^{(2)}_{\text{full}}} E &= \left(\frac{\partial E}{\partial b^{(2)}_{\text{full}}(k)} \right) = \left(\delta^{(2)}_{\text{full}}(k) \right) = \delta^{(2)}_{\text{full}} \in \mathbb{R}^{c_4}\end{aligned}$$

を得る．

したがって，まだ計算が完了していない部分は $\delta^{(2)}_{\text{full}}$ の計算である．これは次のようにされる．

■$\delta^{(2)}_{\text{full}}$ **の計算**

$$\begin{aligned}\delta^{(2)}_{\text{full}}(k) &= \frac{\partial E}{\partial u^{(2)}_{\text{full}}(k)} = \frac{\partial}{\partial u^{(2)}_{\text{full}}(k)}\left(-\sum_{i=1}^{c_4} r(i)\log\left(v^{(2)}(i)\right)\right)\\ &= -r(k) + v^{(2)}(k),\end{aligned}$$

$$\begin{aligned}&\frac{\partial}{\partial u^{(2)}_{\text{full}}(k)}\left(-\sum_{i=1}^{c_4} r(i)\log\left(\text{Soft}(u^{(2)}_{\text{full}})(i)\right)\right)\\ &= -\sum_{i=1}^{c_4} r(i)\frac{\partial}{\partial u^{(2)}_{\text{full}}(k)}\log\left(\frac{\exp(u^{(2)}_{\text{full}}(i))}{\sum_{j=1}^{c_4}\exp(u^{(2)}_{\text{full}}(j))}\right) := (I)\end{aligned}$$

記号を簡略化するために $z_k = u^{(2)}_{\text{full}}(k)$ とおく．$e_{\mu\nu} = \begin{cases} 1, & \mu = \nu \\ 0, & \mu \neq \nu \end{cases}$ とおき，$S = \sum_{j=1}^{c_4}\exp(z_j)$ とおく．

$\frac{\partial S}{\partial z_k} = \exp(z_k)$ である．ゆえに

$$\begin{aligned}(I) &= -\sum_{i=1}^{c_4} r(i)\frac{\partial}{\partial z_k}\log\left(\frac{\exp(z_i)}{S}\right)\\ &= -\sum_{i=1}^{c_4} r(i)\frac{S}{\exp(z_i)}\frac{\frac{\partial\exp(z_i)}{\partial z_k}S - \exp(z_i)\frac{\partial}{\partial z_k}S}{S^2}\\ &= -\sum_{i=1}^{c_4} r(i)\frac{1}{\exp(z_i)}\frac{e_{ik}\exp(z_i)S - \exp(z_i)\exp(z_k)}{S}\\ &= -\sum_{i=1}^{c_4} r(i)\frac{e_{ik}S - \exp(z_k)}{S}\\ &= -r(k)\frac{S-\exp(z_k)}{S} + \sum_{\substack{i=1\\ i\neq k}}^{c_4} r(i)\frac{\exp(z_k)}{S}\\ &= -r(k) + r(k)\frac{\exp(z_k)}{S} + \frac{\exp(z_k)}{S}\sum_{\substack{i=1\\ i\neq k}}^{c_4} r(i)\\ &= -r(k) + v^{(2)}(k)\end{aligned}$$

ここで，最後の式変形は $\sum_{i=1}^{c_4} r(i) = 1$ による．ゆえに

$$\delta^{(2)}_{\text{full}} = -r + v^{(2)} \in \mathbb{R}^{c_4}.$$

以上で第 4 層の勾配ベクトル $\nabla_{W^{(2)}}E$, $\nabla_{b^{(2)}_{\text{full}}}E$ の計算が終了した．

■**第 3 層からの逆伝播** $k = 1, \cdots, c_3$, $i = 1, \cdots, n_2$ として

$$\frac{\partial E}{\partial W^{(1)}(k,i)} = \sum_{j=1}^{c_3}\frac{\partial E}{\partial u^{(1)}_{\text{full}}(j)}\frac{\partial u^{(1)}_{\text{full}}(j)}{\partial W^{(3)}_{\text{full}}(k,i)}.$$

ここで

$$\delta^{(1)}_{\text{full}}(j) = \frac{\partial E}{\partial u^{(1)}_{\text{full}}(j)}$$

とおくと,

$$\frac{\partial E}{\partial W^{(1)}(k,i)} = \sum_{j=1}^{c_3} \delta_{\text{full}}^{(1)}(j) \frac{\partial}{\partial W^{(1)}(k,i)} \left(\sum_{j'=1}^{n_2} W^{(1)}(j,j') v(j') + b_{\text{full}}^{(1)}(j) \right)$$
$$= \delta_{\text{full}}^{(1)}(k) v(i).$$

また

$$\frac{\partial E}{\partial b_{\text{full}}^{(1)}(k)} = \delta_{\text{full}}^{(1)}(k).$$

したがって,

$$\nabla_{W^{(1)}} E = \left(\frac{\partial E}{\partial W^{(1)}(k,i)} \right) = \left(\delta_{\text{full}}^{(1)}(k) v(i) \right) = \delta_{\text{full}}^{(1)} v^T \in \mathbb{R}^{c_3 \times N_2}$$
$$\nabla_{b_{\text{full}}^{(1)}} E = \left(\frac{\partial E}{\partial b_{\text{full}}^{(1)}(k)} \right) = \left(\delta_{\text{full}}^{(1)}(k) \right) = \delta_{\text{full}}^{(1)} \in \mathbb{R}^{c_3}.$$

以下では $\delta_{\text{full}}^{(1)}$ の計算をする.

■$\delta_{\text{full}}^{(1)}$ の計算

$$\delta_{\text{full}}^{(1)}(k) = \frac{\partial E}{\partial u_{\text{full}}^{(1)}(k)} = \sum_{j=1}^{c_4} \frac{\partial E}{\partial u_{\text{full}}^{(2)}(j)} \frac{\partial u_{\text{full}}^{(2)}(j)}{\partial u_{\text{full}}^{(1)}(k)}$$
$$= \sum_{j=1}^{c_4} \delta_{\text{full}}^{(2)}(j) \frac{\partial}{\partial u_{\text{full}}^{(1)}(k)} \left(\sum_{j'=1}^{c_3} W^{(2)}(j,j') v^{(1)}(j') + b_{\text{full}}^{(2)}(j) \right)$$
$$= \sum_{j=1}^{c_4} \delta_{\text{full}}^{(2)}(j) \frac{\partial}{\partial u_{\text{full}}^{(1)}(k)} \left(\sum_{j'=1}^{c_3} W^{(2)}(j,j') f(u_{\text{full}}^{(1)}(j')) + b_{\text{full}}^{(2)}(j) \right)$$
$$= \sum_{j=1}^{c_4} \delta_{\text{full}}^{(2)}(j) W^{(2)}(j,k) f'(u_{\text{full}}^{(1)}(k)).$$

ここで

$$f'(u_{\text{full}}^{(1)}(k)) = \begin{cases} 1, & u_{\text{full}}^{(1)}(k) > 0 \\ 0, & u_{\text{full}}^{(1)}(k) \leq 0 \end{cases}.$$

以上より MATLAB の記法を用いれば

$$\delta_{\text{full}}^{(1)} = \left(\delta_{\text{full}}^{(1)}(k) \right) = \left(u_{\text{full}}^{(1)} > 0 \right)_{\cdot} * W^{(2)T} \delta_{\text{full}}^{(2)} \in \mathbb{R}^{c_3}.$$

と表せる.（注：$W^{(2)} \in \mathbb{R}^{c_4 \times c_3}$ より $W^{(2)T} \in \mathbb{R}^{c_3 \times c_4}$. $\delta_{\text{full}}^{(2)} \in \mathbb{R}^{c_4} = \mathbb{R}^{c_4 \times 1}$ であるから，$W^{(2)T} \delta_{\text{full}}^{(2)} \in \mathbb{R}^{c_3}$.）

以上で $\nabla_{W^{(1)}} E$ と $\nabla_{b_{\text{full}}^{(1)}} E$ の計算が終了した.

■第 2 層からの逆伝播 $k = 1, \cdots, c_2$（第 2 層のフィルタ数），$c = 1, \cdots, c_1$(チャネル)，$\mu = 1, \cdots, m^{(2)}$, $\nu = 1, \cdots, m^{(2)}$ （フィルタサイズ）であることを想起しておく.

次の計算をする.

$$\frac{\partial E}{\partial F^{(2)}(k,c,\mu,\nu)} = \sum_{i=1}^{h_1'} \sum_{j=1}^{w_1'} \frac{\partial E}{\partial u_{\text{conv}}^{(2)}(k,i,j)} \frac{\partial u_{\text{conv}}^{(2)}(k,i,j)}{\partial F^{(2)}(k,c,\mu,\nu)}$$

ここで，h_1', w_1' はプーリングする前のサイズで，$h_2 = \mathrm{fix}(h_1'/2)$, $w_1 = \mathrm{fix}(w_1'/2)$ であったことを思い出しておく．

$$\delta^{(2)}_{\mathrm{conv}}(k,i,j) = \frac{\partial E}{\partial u^{(2)}_{\mathrm{conv}}(k,i,j)}$$

とおく．このとき

$$\begin{aligned}
&\frac{\partial E}{\partial F^{(2)}(k,c,\mu,\nu)} \\
&= \sum_{i=1}^{h_1'}\sum_{j=1}^{w_1'} \delta^{(2)}_{\mathrm{conv}}(k,i,j) \times \\
&\times \frac{\partial}{\partial W^{(2)}(k,c,\mu,\nu)} \left(\sum_{c'=1}^{c_1}\sum_{\mu'=1}^{m^{(2)}}\sum_{\nu'=1}^{m^{(2)}} W^{(2)}(k,c',\mu',\nu') x^{(1)}_{\mathrm{conv}}(c', i+\mu'-1, j+\nu'-1) + b^{(2)}_{\mathrm{conv}}(k) \right) \\
&= \sum_{i=1}^{h_1'}\sum_{j=1}^{w_1'} \delta^{(2)}_{\mathrm{conv}}(k,i,j) x^{(1)}_{\mathrm{conv}}(c, i+\mu-1, j+\nu-1) \\
&= \left(\delta^{(2)}_{\mathrm{conv}}(k,\cdot,\cdot) \bigstar x^{(1)}_{\mathrm{conv}}(c,\cdot,\cdot) \right)(\mu,\nu) .
\end{aligned}$$

また，

$$\frac{\partial E}{\partial b^{(2)}_{\mathrm{conv}}(k)} = \sum_{i=1}^{h_1'}\sum_{j=1}^{w_1'} \delta^{(2)}_{\mathrm{conv}}(k,i,j).$$

【逆プーリング - 全結合層 → 畳み込み層】

ここでプーリングに関する注意をしておく．

$\delta^{(2)}_{\mathrm{conv}}(k,i,j) = \frac{\partial E}{\partial u^{(2)}_{\mathrm{conv}}(k,i,j)}$ は $u^{(2)}_{\mathrm{conv}}(k,i,j)$ に関する偏微分であるが，順伝播では $a^{(2)}_{\mathrm{conv}}(k,i,j) = f(u^{(2)}_{\mathrm{conv}}(k,i,j))$ において，最大プーリングで消えているものと，生き残っているものがある．$i' = 1, \cdots, h_2 = \mathrm{fix}(h_1'/2)$, $j' = 1, \cdots, w_2 = \mathrm{fix}(w_1'/2)$ とする．このとき，プーリングで生き残った位置座標は $\left(\zeta^{(2)}_{(i',j')}, \eta^{(2)}_{(i',j')}\right)$ としていたから，最大プーリングの取り方から

$$\begin{aligned}
1 \le 2i' - 1 \le \zeta^{(2)}_{(i',j')} \le 2i' \le h_1', \\
1 \le 2j' - 1 \le \eta^{(2)}_{(i',j')} \le 2j' \le w_1'
\end{aligned}$$

である．ここで生き残った位置座標以外（つまり生き残らなかったものの位置座標）については $\delta^{(2)}_{\mathrm{conv}}(k,i,j) = 0$ とする．

したがって，実際は生き残らなかったものの箇所は消去して，

$$\begin{aligned}
\frac{\partial E}{\partial F^{(2)}(k,c,\mu,\nu)} &= \sum_{i=1}^{h_1'}\sum_{j=1}^{w_1'} \delta^{(2)}_{\mathrm{conv}}(k,i,j) x^{(1)}_{\mathrm{conv}}(c, i+\mu-1, j+\nu-1) \\
&= \sum_{i'=1}^{h_2}\sum_{j'=1}^{w_2} \delta^{(2)}_{\mathrm{conv}}(k, \zeta^{(2)}_{(i',j')}, \eta^{(2)}_{(i',j')}) x^{(1)}_{\mathrm{conv}}(c, i+\zeta^{(2)}_{(i',j')}-1, j+\eta^{(2)}_{(i',j')}-1), \\
\frac{\partial E}{\partial b^{(2)}_{\mathrm{conv}}(k)} &= \sum_{i'=1}^{h_2}\sum_{j'=1}^{w_2} \delta^{(2)}_{\mathrm{conv}}(k, \zeta^{(2)}_{(i',j')}, \eta^{(2)}_{(i',j')})
\end{aligned}$$

となっている．

以下，生き残った $\delta^{(2)}_{\mathrm{conv}}(k, \zeta^{(2)}_{(i',j')}, \eta^{(2)}_{(i',j')})$ の計算をする．

$k=1,\cdots,c_2$，$i=1,\cdots,h_2$，$j=1,\cdots,w_2$ とする．

$$\begin{aligned}\delta^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)}) &= \frac{\partial E}{\partial u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})} = \sum_{k'=1}^{c_3}\frac{\partial E}{\partial u^{(1)}_{\mathrm{full}}(k')}\frac{\partial u^{(1)}_{\mathrm{full}}(k')}{\partial u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})}\\ &= \sum_{k'=1}^{c_3}\delta^{(1)}_{\mathrm{full}}(k')\frac{\partial}{\partial u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})}\left(\sum_{i'=1}^{n_2}W^{(1)}(k',i')v(i')+b^{(1)}_{\mathrm{full}}(k')\right).\end{aligned}$$

ここで，$n_2 = c_2h_2w_2$ であり，ベクトル化の仕方から，$v(i')$ に対応する並べ替えのもとになった $u^{(2)}_{\mathrm{conv}}(c',h',w')$ が存在する．ここで

$$1\le c'\le c_2,\ 1\le h'\le h_2,\ 1\le w'\le w_2$$

である．$i'=\langle c',h',w'\rangle$ と表す．

$$\begin{aligned}&\delta^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})\\ &= \sum_{k'=1}^{c_3}\delta^{(1)}_{\mathrm{full}}(k')\frac{\partial}{\partial u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})}\left(\sum_{c'=1}^{c_2}\sum_{h'=1}^{h_2}\sum_{w'=1}^{w_2}W^{(1)}(k',\langle c',h',w'\rangle)f(u^{(2)}_{\mathrm{conv}}(c',h',w'))\right)\\ &= \sum_{k'=1}^{c_3}\delta^{(1)}_{\mathrm{full}}(k')W^{(1)}\left(k',\left\langle k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)}\right\rangle\right)\frac{\partial f(u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)}))}{\partial u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)})}\\ &= f'(u^{(2)}_{\mathrm{conv}}(k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)}))\sum_{k'=1}^{c_3}\delta^{(1)}_{\mathrm{full}}(k')W^{(1)}\left(k',\left\langle k,\zeta^{(2)}_{(i,j)},\eta^{(2)}_{(i,j)}\right\rangle\right).\end{aligned}$$

以上のプーリングに関する項について注意をしたうえで

$$\begin{aligned}\nabla_{F^{(2)}}E &= \left(\frac{\partial E}{\partial F^{(2)}(k,c,\mu,\nu)}\right)\in\mathbb{R}^{c_2\times c_1\times m^{(2)}\times m^{(2)}}\\ \nabla_{\delta^{(2)}_{\mathrm{conv}}}E &= \left(\delta^{(2)}_{\mathrm{conv}}(k,i,j)\right)\in\mathbb{R}^{c_2\times h_1'\times w_1'}\end{aligned}$$

の計算が終了した．

■第 1 層からの逆伝播　$k=1,\cdots,c_1$ （フィルタ数），$c=1,\cdots,c_0$ （チャネル数），$\mu=1,\cdots,m^{(1)}$，$\nu=1,\cdots,m^{(1)}$ （フィルタ・サイズ）であることを想起しておく．

$$\begin{aligned}\frac{\partial E}{\partial F^{(1)}(k,c,\mu,\nu)} &= \sum_{i=1}^{h_0}\sum_{j=1}^{w_0}\frac{\partial E}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)}\frac{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)}{\partial F^{(1)}(k,c,\mu,\nu)},\\ \nabla_{F^{(1)}}E &= \left(\frac{\partial E}{\partial F^{(1)}(k,c,\mu,\nu)}\right)\in\mathbb{R}^{c_1\times c_0\times m^{(1)}\times m^{(1)}}.\\ \nabla_{b^{(1)}_{\mathrm{conv}}}E &= \left(\frac{\partial E}{\partial b^{(1)}_{\mathrm{conv}}(k)}\right)\in\mathbb{R}^{c_1}.\end{aligned}$$

ここで

$$\begin{aligned}\delta^{(1)}_{\mathrm{conv}}(k,i,j) &= \frac{\partial E}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)},\\ \delta^{(1)}_{\mathrm{conv}} &= \left(\delta^{(1)}_{\mathrm{conv}}(k,i,j)\right)\in\mathbb{R}^{c_1\times h_0\times w_0}\end{aligned}$$

とおく．このとき

$$
\begin{aligned}
&\frac{\partial E}{\partial F^{(1)}(k,c,\mu,\nu)}\\
&=\sum_{i=1}^{h_0'}\sum_{j=1}^{w_0'}\delta^{(1)}_{\mathrm{conv}}(k,i,j)\times\\
&\frac{\partial}{\partial F^{(1)}(k,c,\mu,\nu)}\left(\sum_{c'=1}^{c_0}\sum_{\mu'=1}^{m^{(1)}}\sum_{\nu'=1}^{m^{(1)}}F^{(1)}(k,c',\mu',\nu')x^{(0)}(c',i+\mu'-1,j+\nu'-1)+b^{(1)}_{\mathrm{conv}}(k)\right)\\
&=\sum_{i=1}^{h_0'}\sum_{j=1}^{w_0'}\delta^{(1)}_{\mathrm{conv}}(k,i,j)x^{(0)}(c,i+\mu-1,j+\nu-1)\\
&=\left(\delta^{(1)}_{\mathrm{conv}}(k,\cdot,\cdot)\bigstar x^{(0)}(c.\cdot,\cdot)\right)(\mu,\nu).
\end{aligned}
$$

また

$$
\frac{\partial E}{\partial b^{(1)}_{\mathrm{conv}}(k)}=\sum_{i=1}^{h_0'}\sum_{j=1}^{w_0'}\delta^{(1)}_{\mathrm{conv}}(k,i,j).
$$

【逆プーリング - 畳み込み層 → 畳み込み層】

プーリングに関する注意をしておく．

$\delta^{(1)}_{\mathrm{conv}}(k,i,j)=\frac{\partial E}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)}$ であるが，$a^{(1)}_{\mathrm{conv}}(k,i,j)=f(u^{(1)}_{\mathrm{conv}}(k,i,j))$ を最大プーリングしている．これにより切り捨てられた (k,i,j) に対しては $\delta^{(1)}(k,i,j)=0$ と定める．残したところを考える．

$$
i'=1,\cdots,h_1=\mathrm{fix}(h_0'/2),\ j'=1,\cdots,w_1=\mathrm{fix}(w_0'/2)
$$

とする．このとき，プーリングで生き残った位置座標は $\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')}$ であったから，最大プーリングの取り方から

$$
\begin{aligned}
&1\le 2i'-1\le\zeta^{(1)}_{(i',j')}\le 2i'\le h_0',\\
&1\le 2j'-1\le\eta^{(1)}_{(i',j')}\le 2j'\le w_0'.
\end{aligned}
$$

いま

$$
\begin{aligned}
&\frac{\partial E}{\partial F^{(1)}(k,c,\mu,\nu)}\\
&=\sum_{i=1}^{h_0'}\sum_{j=1}^{w_0'}\delta^{(1)}_{\mathrm{conv}}(k,i,j)x^{(0)}(c,i+\mu-1,j+\nu-1)\\
&=\sum_{i'=1}^{h_1}\sum_{j'=1}^{w_1}\delta^{(1)}_{\mathrm{conv}}(k,\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')})x^{(0)}(c,\zeta^{(1)}_{(i',j')}+\mu-1,\eta^{(1)}_{(i',j')}+\nu-1),
\end{aligned}
$$

$$
\frac{\partial E}{\partial b^{(1)}_{\mathrm{conv}}(k)}=\sum_{i=1}^{h_0'}\sum_{j=1}^{w_0'}\delta^{(1)}_{\mathrm{conv}}(k,i,j)=\sum_{i'=1}^{h_1}\sum_{j'=1}^{w_1}\delta^{(1)}_{\mathrm{conv}}(k,\zeta_{i'},\eta_{j'}).
$$

さて $\delta^{(1)}_{\mathrm{conv}}(k,\zeta^{(1)}_{(i',j')},\eta^{(1)}_{(i',j')})$ の計算をする．ここで，$\zeta^{(1)}_{(i',j')}=i$，$\eta^{(1)}_{(i',j')}=j$ とおくとき，$\alpha^{(1)}_{(i,j)}=i'$，

$\beta^{(1)}_{(i,j)} = j'$ と定めた（既述の記号）.

$$
\begin{aligned}
&\delta^{(1)}_{\mathrm{conv}}\left(k, \zeta^{(1)}_{(i',j')}, \eta^{(1)}_{(i',j')}\right) \\
&= \delta^{(1)}_{\mathrm{conv}}(k,i,j) = \frac{\partial E}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)} \\
&= \sum_{c=1}^{c_2}\sum_{\mu=1}^{m^{(2)}}\sum_{\nu=1}^{m^{(2)}} \frac{\partial E}{\partial u^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1)} \frac{\partial u^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1)}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)} \\
&= \sum_{c=1}^{c_2}\sum_{\mu=1}^{m^{(2)}}\sum_{\nu=1}^{m^{(2)}} \delta^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1) \frac{\partial u^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1)}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)}.
\end{aligned}
$$

ここで

$$
\begin{aligned}
&u^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1) \\
&= \sum_{c'=1}^{c_1}\sum_{\mu'=1}^{m^{(2)}}\sum_{\nu'=1}^{m^{(2)}} F^{(2)}(c,c',\mu',\nu') x^{(1)}_{\mathrm{conv}}(c', \alpha^{(1)}_{(i,j)} - \mu + 1 + \mu' - 1, \beta^{(1)}_{(i,j)} - \nu + 1 + \nu' - 1) \\
&+ b^{(2)}_{\mathrm{conv}}(c) \\
&= \sum_{c'=1}^{c_1}\sum_{\mu'=1}^{m^{(2)}}\sum_{\nu'=1}^{m^{(2)}} W^{(2)}(c,c',\mu',\nu') x^{(1)}_{\mathrm{conv}}(c', \alpha^{(1)}_{(i,j)} - \mu + \mu', \beta^{(1)}_{(i,j)} - \nu + \nu') + b^{(2)}_{\mathrm{conv}}(c)
\end{aligned}
$$

そこで，$x^{(1)}(c, \alpha^{(1)}_{(i,j)}, \beta^{(1)}_{(i,j)}) = f(u^{(1)}(c,i,j))$ に注意すると

$$
\frac{\partial u^{(2)}_{\mathrm{conv}}(c, p_i - \mu + 1, q_j - \nu + 1)}{\partial u^{(1)}_{\mathrm{conv}}(k,i,j)} = F^{(2)}(c,k,\mu,\nu) f'(u^{(1)}_{\mathrm{conv}}(k,i,j))
$$

が得られる．ゆえに

$$
\begin{aligned}
\delta^{(1)}_{\mathrm{conv}}(k,i,j) &= \sum_{c=1}^{c_2}\sum_{\mu=1}^{m^{(2)}}\sum_{\nu=1}^{m^{(2)}} \delta^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1) F^{(2)}(c,k,\mu,\nu) f'(u^{(1)}(k,i,j)) \\
&= f'(u^{(1)}_{\mathrm{conv}}(k,i,j)) \sum_{c=1}^{c_2}\sum_{\mu=1}^{m^{(2)}}\sum_{\nu=1}^{m^{(2)}} \delta^{(2)}_{\mathrm{conv}}(c, \alpha^{(1)}_{(i,j)} - \mu + 1, \beta^{(1)}_{(i,j)} - \nu + 1) F^{(2)}(c,k,\mu,\nu)
\end{aligned}
$$

以上で $\frac{\partial E}{\partial F^{(1)}(k,c,\mu,\nu)}, \frac{\partial E}{\partial b^{(1)}_{\mathrm{conv}}(k)}$ の計算ができた.

3.4 パラメータの更新

勾配降下法

すでに述べたように，$\varepsilon > 0$ を学習率として

$$
\begin{aligned}
F^{(1)} - \varepsilon \nabla_{F^{(1)}} E &\to F^{(1)}, \\
F^{(2)} - \varepsilon \nabla_{F^{(2)}} E &\to F^{(2)}, \\
W^{(1)} - \varepsilon \nabla_{W^{(1)}} E &\to W^{(1)}, \\
W^{(2)} - \varepsilon \nabla_{W^{(2)}} E &\to W^{(2)}.
\end{aligned}
$$

$$
\begin{aligned}
b^{(1)}_{\mathrm{conv}} - \alpha\nabla_{b^{(1)}_{\mathrm{conv}}} E &\to b^{(1)}_{\mathrm{conv}},\\
b^{(2)}_{\mathrm{conv}} - \alpha\nabla_{b^{(2)}_{\mathrm{conv}}} E &\to b^{(2)}_{\mathrm{conv}},\\
b^{(1)}_{\mathrm{full}} - \alpha\nabla_{b^{(1)}_{\mathrm{full}}} E &\to b^{(1)}_{\mathrm{full}},\\
b^{(2)}_{\mathrm{full}} - \alpha\nabla_{b^{(2)}_{\mathrm{full}}} E &\to b^{(2)}_{\mathrm{full}}.
\end{aligned}
$$

のようにフィルタ/重み，バイアスを更新する．

参考文献

[1] 新井仁之，プログラミングしながら学ぶ深層学習 1 - 数学の解説から分類のプログラミングまでを MATLAB で解説，http://www.araiweb.matrix.jp/Program/MLP1.html

[2] 新井仁之，プログラミングしながら学ぶ応用線形代数と深層学習，http://www.araiweb.matrix.jp/Linear.html

[3] 新井仁之，これからの微分積分，日本評論社，2019.

[4] 我妻幸長，はじめてのディープラーニング - Python で学ぶニューラルネットワークとバックプロパゲーション，SB Creative, 2018.

[5] 岡谷貴之，深層学習，改訂第 2 版，講談社，2022.

[6] 立石賢吾，やさしく学ぶディープラーニングがわかる数学のきほん，マイナビ，2019.

Appendix　MATLAB プログラム

畳み込みニューラルネットの深層学習：MNIST 手書き文字分類

MATLAB コードを載せておく．なお MATLAB コードは，本稿では説明を省いたが，モメンタム法とドロップアウトを採用してある．

なお教育目的から MATLAB の Deep Learning Toolbox は使用していない．

適宜，更新版・改訂版を下記サイトに掲載する．

www.araiweb.matrix.jp/DeepLearning.html

(*) 本文と Appendix の version は独立してます．

畳み込みニューラルネットの深層学習：MNIST 手書き数字の分類（MATLAB）

新井仁之（早稲田大学）

Ver.1.3. 2025 年 1 月 12 日　文章追加

Ver.1.2 2025 年 1 月 6 日　文章追加

Ver. 1.1 2025 年 1 月 5 日

標記のプログラムを MATLAB で書いたものである．本プログラムを書くに当たっては，Appendix の末尾にある「プログラムの参考文献」に記載の [1] ～ [4] （特に [3]） の Python コードに基づいて作成したが，ただしここでは MATLAB のセル構造を使い層を自由に増やせるようにした．また，モメンタム法を採用し，ドロップアウトも付け加えるようにした．教育上の目的から，Deep learning toolbox は用いていない．

なお，畳み込み Cnv2，Im2Col，Col2Im，最大プーリング，逆プーリングについては，詳しいチュートリアルも書いたので，

http://www.araiweb.matrix.jp/Linear.html

を必要に応じて参照してほしい．

以下では，畳み込みは same 型である．

```
clear
clc
```

MNIST イメージ読み込み

MNIST 手書き文字のデータファイルをダウンロードし，MATLAB のワークスペースに保存しておく．この部分は各自 MNIST の公式ページ https://yann.lecun.com/exdb/mnist/ からデータをダウンローしてほしい．

＊）ここでは予め TrainImages，TrainLabels，TestImages，TestLabels をダウンロードし mat ファイルにした。

```
% CombinedFile.mat を読み込む
combinedData = load('MNIST.mat');

% 統合されたデータのアクセス例
TrainImages = combinedData.TrainImages_file1;
```

```
TrainLabels = combinedData.TrainLabels_file2;
TestImages = combinedData.TestImages_file3;
TestLabels = combinedData.TestLabels_file4;
```

各データのサイズ

```
whos TrainImages
```

```
  Name              Size                       Bytes  Class     Attributes

  TrainImages       28x28x1x60000          376320000  double
```

```
whos TrainLabels
```

```
  Name              Size                Bytes  Class     Attributes

  TrainLabels       1x60000            480000  double
```

```
whos TestImages
```

```
  Name              Size                       Bytes  Class     Attributes

  TestImages        28x28x1x10000           62720000  double
```

```
whos TestLabels
```

```
  Name              Size                Bytes  Class     Attributes

  TestLabels        1x10000             80000  double
```

以上の準備のもとに畳み込みニューラルネットの深層学習による手書き手書き文字画像の分類を始める.

```
TrainImages = permute(TrainImages,[4 3 1 2]);
size(TrainImages)
```

```
ans = 1×4
       60000           1          28          28
```

学習用画像データに対する正解データ

```
%%%%
% 正解データをベクトル表示
% 例えば正解が 3 なら 0 0 1 0 0 0 0 0 0 0 の形に変更
%%%%
TrainLabels(TrainLabels == 0) = 10;
TrainLabels = permute(TrainLabels,[2 1]);
TrainLabels_vector=zeros(60000,10);
for i=1:60000
    k=TrainLabels(i,1);
    TrainLabels_vector(i,k)=1;
end
```

```
size(TrainLabels_vector)
```

```
ans = 1×2
       60000          10
```

```
TrainLabels_vector(1:3,:)
```

```
ans = 3×10
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1
     0     0     0     1     0     0     0     0     0     0
```

畳み込みニューラルネットの基本設計図の設定

```
%入力画像サイズの指定
init_image_size = [1,28,28]; % [チャネル数，縦サイズ 横サイ
ズ]
                             % [1,28,28] は MNIST 手書き文字データのものである．

% 層の数の指定
L1 = 2; %畳み込み層の数の指定 画像サイズとプーリングのため，L1=1 または 2 に限る．
L2 = 3; %全結合層の数の指定

% 各層のフィルタ数の設定
n1 = [32,64]; %各畳み込み層(1～L1 層)のフィルタ数の指定

% 各層のウェイト数の設定
n2 = [100 200 10]; %各全結合層（L1+1~L1+L2）のユニットの数の指定
```

学習率，モメンタム法パラメータ，ドロップアウト率，エポック数，ミニバッチサイズの設定

```
% 学習率
Eta = 0.0001;
% モメンタム法．Beta=0 にすれば通常の方法．
% Beta = -0.2;
Beta = -0.2;
% ドロップアウト率
drop_ratio = 0.2;
% エポック数の設定
Epoch = 4;
% ミニバッチサイズの設定
BatchSize = 100;
```

フィルタ，重み，バイアスの格納セルの設定

```
%%% 畳み込み層のフィルタとバイアスの格納セル
Filters = {1,L1}; %セル
```

```
bias_c= {1,L1};   %セル
%%% 全結合層の重みとバイアスの格納セル
Weights = {1,L2}; %セル
bias_t = {1,L2};  %セル
```

```
% 入力層のチャネル数を追加
n1 = [init_image_size(1),n1];
```

```
%最初の全結合層の入力データの次元の計算；ただし畳み込みは 'same' 型
M = fix(init_image_size(2)/2^L1);
M = n1(L1+1)*(M^2);
n3 = [M,n2];
Setting = {L1,L2,n1,n3}; %読み込み用パラメータ
```

畳み込み用フィルタをランダムに設定

```
FilterSize = [3,3]; %フィルタサイズの指定（全フィルタ共通）
FT = FilterSize(1)*FilterSize(2);
for k =1:L1
    %He による正規化
    Filters{1,k} = randn(n1(k+1),n1(k),FilterSize(1),FilterSize(2))*sqrt(2/
(n1(k)*FT));
    bias_c{1,k} = zeros(1,size(Filters{1,k},1));
end
```

全結合層の重みとバイアスをランダムに設定する場合（He の方法）

```
for k = 1:L2
    Weights{1,k} = randn(n3(k+1),n3(k))*sqrt(2/(n3(k)));
    bias_t{1,k} = zeros(1,size(Weights{1,k},1));
end
```

初期モーメントの設定（ゼロに設定）

```
Moment_c = {1,L1};
Moment_t = {1,L2};
biasMoment_c = {1,L1};
biasMoment_t = {1,L2};
%
for k = 1:L1
    Moment_c{1,k} = zeros(size(Filters{1,k}));
    biasMoment_c{1,k} = zeros(size(bias_c{1,k}));
end
for k = 1:L2
    Moment_t{1,k} =zeros(size(Weights{1,k}));
    biasMoment_t{1,k} = zeros(size(bias_t{1,k}));
end
```

順処理と誤差逆伝播法

以下では次の**出力用データ蓄積用セル**が作業上で設定される.

畳み込み層

Z_c = {1,L1}; アフィン変換によるデータ用

X_c = {1,L1}; 活性化によるデータ用

PI = {1,L1}; 最大プーリングの最大値の位置情報用

D_c = {1,L1}; 誤差逆伝播のデルタ

全結合層

Z_t = {1,L2}; アフィン変換によるデータ用

X_t = {1,L2}; 活性化によるデータ用

D_t = {1,L2}; 誤差逆伝播のデルタ

```
tic % 時間がかかる CPU

for epoch = 1:Epoch
    disp(['Epoch = ',num2str(epoch)]);
    p = randperm(length(TrainImages));
    %p = 1: length(TrainImages); %ランダムに配列しなおさない場合
    for i = 1:ceil(length(TrainImages)/BatchSize)
        indice = p((i-1)*BatchSize+1: i*BatchSize);
        X0 = TrainImages(indice,1,:,:);
        Y = TrainLabels_vector(indice,:);
        % 順計算
        [Z_c, X_c, PI, Z_t, X_t] = Forward(X0,Filters,Weights,
bias_c,bias_t,Setting,drop_ratio);
        % 誤差逆伝播法のデルタの計算
        [D_c,D_t] = Backward(Y,X_c,Z_c,X_t,Z_t,PI,Filters,Weights,Setting);
        % フィルタ，重みの更新（モーメント法）

[Filters,Weights,bias_c,bias_t,Moment_c,biasMoment_c,Moment_t,biasMoment_t]=Update_P
arameters_Momentum3(D_t, ...
            D_c,X_t,X_c,X0,Filters,Weights,bias_c,bias_t,Eta,
Beta,Moment_t,Moment_c,biasMoment_t,biasMoment_c,Setting);

        if mod(i,10)==0
            Error = sum(-Y.*log(X_t{1,L2}+1e-5),'all') %誤差表示
            %Error = sum(abs(-Y+X_t{1,L2}),'all') %誤差表示
        end
    end
end
```

```
Epoch = 1
Error = 187.0938
Error = 154.2223
```

```
Error = 117.5135
Error = 107.3014
Error = 101.3021
Error = 69.7032
Error = 60.1384
Error = 80.3276
Error = 71.5251
Error = 62.1540
Error = 76.2987
Error = 57.6518
Error = 45.5254
Error = 74.5437
Error = 64.2208
Error = 58.5835
Error = 48.9222
Error = 41.6133
Error = 58.5802
Error = 44.5758
Error = 51.5016
Error = 44.2006
Error = 51.9910
Error = 38.2157
Error = 53.8233
Error = 48.7570
Error = 37.7552
Error = 27.3906
Error = 50.5307
Error = 33.1821
Error = 26.9751
Error = 22.9181
Error = 37.8468
Error = 52.7764
Error = 32.8974
Error = 52.4852
Error = 46.7627
Error = 39.0996
Error = 41.5830
Error = 34.5535
Error = 42.3832
Error = 40.3643
Error = 36.7090
Error = 40.7426
Error = 30.4544
Error = 41.6558
Error = 35.8866
Error = 49.3361
Error = 51.0127
Error = 28.0032
Error = 23.8644
Error = 16.4526
Error = 52.4258
Error = 26.5160
Error = 45.3697
Error = 30.7617
Error = 19.3094
Error = 26.6900
Error = 47.1224
Error = 27.8648
Epoch = 2
Error = 17.9712
Error = 19.8456
Error = 42.3838
Error = 18.7059
Error = 20.4427
```

```
Error = 29.8678
Error = 27.5733
Error = 25.4239
Error = 24.2062
Error = 46.4912
Error = 42.9663
Error = 32.7472
Error = 32.4016
Error = 23.7074
Error = 16.6487
Error = 17.2878
Error = 22.2105
Error = 27.7210
Error = 43.5128
Error = 21.5606
Error = 19.6045
Error = 19.6218
Error = 22.8724
Error = 17.9338
Error = 15.4357
Error = 17.0479
Error = 15.5933
Error = 29.9307
Error = 21.4913
Error = 41.7859
Error = 40.5609
Error = 17.0601
Error = 27.4634
Error = 14.2217
Error = 20.1364
Error = 28.3615
Error = 28.9792
Error = 27.1420
Error = 20.4150
Error = 14.3001
Error = 29.4409
Error = 24.7152
Error = 24.1101
Error = 46.0693
Error = 12.1892
Error = 11.7126
Error = 29.5687
Error = 8.6009
Error = 23.2503
Error = 18.8115
Error = 22.9263
Error = 19.9231
Error = 24.2117
Error = 10.6676
Error = 21.7786
Error = 20.9067
Error = 30.4851
Error = 23.5525
Error = 23.8264
Error = 22.1190
Epoch = 3
Error = 19.2768
Error = 29.8607
Error = 24.3771
Error = 31.8694
Error = 24.1968
Error = 24.8173
Error = 26.4573
Error = 22.9752
```

```
Error = 26.0388
Error = 20.6534
Error = 29.1067
Error = 17.2930
Error = 23.3750
Error = 20.0302
Error = 20.2315
Error = 7.5367
Error = 9.7960
Error = 23.1133
Error = 12.8893
Error = 25.5206
Error = 17.3525
Error = 39.4160
Error = 8.2115
Error = 14.3914
Error = 10.8945
Error = 23.4490
Error = 26.7701
Error = 13.8361
Error = 13.9693
Error = 24.7258
Error = 17.8551
Error = 27.9483
Error = 13.0666
Error = 14.3412
Error = 11.7164
Error = 14.7687
Error = 24.8150
Error = 16.2963
Error = 20.0386
Error = 28.0697
Error = 11.0644
Error = 19.8943
Error = 11.4326
Error = 9.2160
Error = 10.1255
Error = 18.8511
Error = 32.0373
Error = 11.2674
Error = 19.2310
Error = 10.6894
Error = 28.6003
Error = 29.0682
Error = 23.5396
Error = 20.3993
Error = 18.8565
Error = 21.3105
Error = 12.4028
Error = 9.2942
Error = 33.1903
Error = 17.0115
Epoch = 4
Error = 20.7610
Error = 19.5488
Error = 6.6304
Error = 24.3450
Error = 8.5416
Error = 29.9209
Error = 12.2441
Error = 16.0289
Error = 25.4931
Error = 8.3173
Error = 10.9692
```

```
Error = 14.6697
Error = 8.3543
Error = 26.0118
Error = 14.9715
Error = 18.9929
Error = 7.7125
Error = 12.8754
Error = 24.4945
Error = 23.5788
Error = 20.3461
Error = 28.1717
Error = 20.7155
Error = 20.4459
Error = 17.9059
Error = 8.5980
Error = 6.1635
Error = 8.0842
Error = 9.9898
Error = 16.8833
Error = 8.4715
Error = 17.6937
Error = 25.1088
Error = 34.2543
Error = 7.6551
Error = 22.4387
Error = 10.7015
Error = 13.5003
Error = 16.6288
Error = 16.7999
Error = 4.2738
Error = 22.1706
Error = 6.7277
Error = 12.8019
Error = 17.5114
Error = 16.6468
Error = 8.1301
Error = 24.8585
Error = 15.6152
Error = 26.5191
Error = 14.7005
Error = 21.1266
Error = 17.8858
Error = 19.2591
Error = 9.9611
Error = 17.9872
Error = 17.9793
Error = 20.3158
Error = 10.1157
Error = 29.8966
```

```
toc
```

経過時間は 805.452498 秒です。

テスト画像で学習結果をテスト

```
% テスト画像の準備
NumTestImages=1000; %テスト画像数 1000
```

```
X = TestImages(:,:,1:NumTestImages); % テスト用画像
X = permute(X,[3 1 2]);
Z = zeros(NumTestImages,1,28,28);
Z(:,1,:,:)=X(:,:,:);
TX=Z;
%size(TX)
TL = TestLabels(1:NumTestImages); % テスト用画像の正解データ
TL(TL == 0) = 10; % 0 は 10 にする
TL = permute(TL,[2 1]);
%size(D)
```

正解率の計算

```
tic;
acc = 0;
N=length(TL);
for i=1:N
    X0test = zeros(1,1,28,28);
    X0test(1,1,:,:)=TX(i,1,:,:);
    [~, ~, ~, ~, X_t] = Forward(X0test,Filters,Weights, bias_c,bias_t,Setting,0);
    [~,j] =max(X_t{1,L2});
    if j==TL(i)
        acc = acc+1;
    end
end
acc = acc/N;
fprintf('正解率は %f\n', acc);
```

```
正解率は 0.971000
```

```
toc;
```

```
経過時間は 1.338477 秒です。
```

テスト画像：最初の 10 枚　画像と推測値の表示

```
X0test = zeros(10,1,28,28);
X0test(1:10,1,:,:)=TX(1:10,1,:,:);
[~, ~, ~, ~, X_t] = Forward(X0test,Filters,Weights, bias_c,bias_t,Setting,0);
TestData = zeros(28,28);
for i=1:10
    TestData(:,:)=X0test(i,1,:,:);
    figure
    imshow(TestData)
    A=X_t{1,L2}(i,:);
    maximum = max(A);
    answer=find(A==maximum);
    answer
```

```
end
```

```
answer = 7
```

```
answer = 2
```

```
answer = 1
```

```
answer = 10
```

```
answer = 4
```

```
answer = 1
```

```
answer = 4
```

```
answer = 9
```

```
answer = 6
```

```
answer = 9
```

このプログラム実行に必要な関数ファイル

Forward　順計算を行う関数ファイル

```
function [Z_c, X_c, PI, Z_t, X_t] = Forward(X0,Filters,Weights,
bias_c,bias_t,Setting,drop_ratio)

L1 = Setting{1};
L2 = Setting{2};

%畳み込み層の計算
[X1,PI1,Z1] = Conv_Layer(X0,Filters{1,1},bias_c{1,1});
X_c{1,1} = X1;
Z_c{1,1} =Z1;
```

```
PI{1,1} = PI1;

for k = 2:L1
    [X,P,Z] = Conv_Layer(X_c{1,k-1},Filters{1,k},bias_c{1,k});
    X_c{1,k} = X;
    Z_c{1,k} = Z;
    PI{1,k} = P;
end

% 畳み込み層の出力を全結合層用に再配列
X_c{1,L1} = reshape_1(X_c{1,L1});

% 全結合層の計算
[X,Z] = Total_Layer(X_c{1,L1}, Weights{1,1},bias_t{1,1},0);
X_t{1,1} = X;
Z_t{1,1} = Z;

if L2 == 1 %全結合層が 1 層のみの場合
    X = SoftMax(Z_t{1,1});
    X_t{1,2} = X; %出力
else
    for k = 2:L2 %全結合層が複数ある場合
        [X,Z] = Total_Layer(X_t{1,k-1}, Weights{1,k},bias_t{1,k},drop_ratio);
        X_t{1,k} = X;
        Z_t{1,k} = Z;
    end
end
X_t{1,L2} = SoftMax(Z_t{1,L2}); %出力

end
```

Conv_Layer

```
function [X,PI,Z] = Conv_Layer(X,W,b)
pad=fix(size(W,3)/2);
Z = Cnv2(X,W,b,1,pad);
A = max(Z,0); % ReLU 関数
[X,PI] = Pooling(A,[2,2],2,0);
end
```

Cnv2 (convolution)

```
function Z = Cnv2(Image,Filter,Bias,Stride,Padding)

% Stride = 1, Padding = 0 -> 'valid'型畳み込み
% Stride = 1, Padding = 1 -> 'same'型畳み込み

%%%%%%%%%%%%%%%%%%%%%%%%%%
% 入力タイプ
% Image = Image(Number of Images, Channel, Image_height, Image_width)
```

```
% Filter = Filter(Number of Filters, Channel, Filter_height, Filter_width)
% Bias = number
% Stride = number
% Padding = 0 または 1
%
% 出力 Z のサイズ:
% Number of Images x Number of Filters x output_hight x output_width
% ここで
% Output_hight = fix((Image_height-Block_height+2*Padding)/Stride)+1
% Output_width = fix((Image_width-Block_width+2*Padding)/Stride)+1
%
% 本ノートの参考文献 [2], [3], [4] の Python プログラムにもとづく.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

[NumImage, ~, Image_height, Image_width] = size(Image);
[NumFilter,Channel,Filter_height,Filter_width] = size(Filter);
Output_height = fix((Image_height -Filter_height +2*Padding)/Stride)+1;
Output_width = fix((Image_width -Filter_width +2*Padding)/Stride)+1;
Image = Im2Col(Image,[Filter_height,Filter_width],Stride,Padding);

% Im2Col が Python 仕様の配列の場合
%Filter = permute(Filter,[4 3 2 1]);

%% Im2Col が  MATLAB 仕様の配列方式の場合
Filter = permute(Filter,[3 4 2 1]);

Filter = reshape(Filter,[Channel*Filter_height*Filter_width NumFilter])';
Filter = permute(Filter,[2 1]);

%Z = X*W+b;
Z = affine_product(Image,Filter,Bias);

Z = permute(Z,[2 1]);
Z = reshape(Z,[NumFilter Output_width Output_height NumImage]);

%% Python 仕様の配列方式の場合
% Z = permute(Z,[4 1 3 2]);

%% MATLAB 仕様の配列方式の場合
Z = permute(Z,[4 1 2 3]);

% フィルタのサイズが偶数の場合は切り取る.
if Padding == 1
    if mod(Filter_height,2) == 0
        Z = Z(:,:,2:Output_height,:);
    end
    if mod(Filter_width,2) == 0
        Z = Z(:,:,:,2:Output_width);
    end
```

```
end
end
```

アフィン変換

```
function C = affine_product(A,B,b)

% 画像 A とファイルタ B，そしてバイアス = 実数とのアフィン積

C=A*B;
k=size(C,1);
ba=repmat(b,[k,1]);
C=C+ba;
end
```

Pooling 最大プーリング

```
function [Pooling_Image,Max_positions] = Pooling(Image,block,Stride,Padding)
% 入力タイプ Images(Number of Images, Number of Channels, Image_height, Image_width)
% 出力タイプ Pooling_image(Number of Images Number of Channels Output_height
Output_width)
%
% [3] の Python コードに基づく.
%

Image = padarray(Image,[0 0 Padding Padding],0,'both');
[NumImage, Channels, Image_height, Image_width] = size(Image);
block_height=block(1);
block_width=block(2);
Output_height = fix((Image_height - block_height + 2*Padding)/Stride)+1;
Output_width = fix((Image_width - block_width + 2*Padding)/Stride)+1;

Col = Im2Col(Image,[block_height,block_width],Stride,Padding);
Col = permute(Col,[2 1]);
Col = reshape(Col,[block_height*block_width
Channels*NumImage*Output_height*Output_width]);

[Max_values,Max_positions] = max(Col);
Pooling_Image = reshape(Max_values,[Channels Output_height Output_width NumImage]);
Pooling_Image = permute(Pooling_Image,[4 1 3 2]);
end
```

Total_Layer (オプション: Dropout)

```
function [Y,Z] = Total_Layer(X,W,b,drop_ratio)
Z = X*(W')+b;
Y = max(Z,0); % ReLU 関数

Y = Y.*Dropout(Y,drop_ratio);
```

```
%Y = Y.*Dropout(X,0.2); %Dropout をするときに追加.
%% 下記の Dropout 関数もコメント解除する.
end
function ym = Dropout(y,ratio)
[m,n] = size(y);
ym = zeros(m,n);
num = round(m*n*(1-ratio));
idx = randperm(m*n,num);
ym(idx)=1/(1-ratio);
end
```

ソフトマックス関数

```
function Y = SoftMax(X)
Y = X - max(X,[],2);
Y = exp(Y)./sum(exp(Y),2);
end
```

Backward

```
function [D_c,D_t] = Backward(Y,X_c,Z_c,X_t,Z_t,PI,Filters,Weights,Setting)
L1 = Setting{1};
L2 = Setting{2};

D_t{1,L2} = -Y+X_t{1,L2}; %最終層のデルタ

for k = L2-1:-1:1
    D_t{1,k} = (Z_t{1,k}>0).*(D_t{1,k+1}*Weights{1,k+1}); %全結合層の第 k 番目の層のデルタ
end

D = (X_c{1,L1}>0).*(D_t{1,1}*Weights{1,1});   %畳み込み層 第 L1 層のデルタ 畳み込み部分
D = Max_unpooling(Z_c{1,L1},PI{L1},D,[2,2],2);%畳み込み層 第 L1 層のデルタ 逆プーリング部分
D_c{1,L1} = D;

for i = L1-1:-1:1
    s = fix(size(Filters{1,i},3)/2);
    D1 = Delta_ConvLayer(X_c{1,i},D_c{1,i+1},Filters{1,i+1},1,s); %畳み込み層のデルタ
    D1 = Max_unpooling(Z_c{1,i},PI{i},D1,[2,2],2);
    D_c{1,i} = D1;
end

end
```

逆最大プーリング

```
function X = Max_unpooling(Image_type,PI,Delta,block,Stride)
%[NumImage,Channel_Images,Image_height,Image_width] = size(Image_type);
%
```

```
% 下記参考文献 [3] の Python プログラムに基づく.
%
D_flat = reshape_1(Delta);

n=size(PI,2);
col_D=zeros(n,4);
for k=1:n
    col_D(k,PI(k))=D_flat(k);
end

X=Col2Im(col_D,Image_type,block,Stride,0);
end
```

Max_unpooling で使う reshape_1 の定義

```
function Y = reshape_1(X)
N=size(X,1);
m2 = permute(X,ndims(X):-1:1); % ndims(X) は X の次元数
m3 = reshape(m2,numel(X)/N,N); % numel(X) は X の要素数
Y = permute(m3,ndims(m3):-1:1);
end
```

Delta_ConvLayer

```
function dcx=Delta_ConvLayer(Output_shape,D,W,Stride,Padding)

% 下記参考文献 [3] の Python プログラムに基づく.
% D1 = delta_conv(X1,D2,W2,1,s); % 本プログラムで使う場合
[Num_D,Channel_D,D_height,D_width]=size(D);
[Num_Filter,Channel_Filter,Filter_height,Filter_width]=size(W);

D = permute(D,[2 3 4 1]); % [Channel_D D_height D_width Num_D]
D = reshape(D,[Channel_D Num_D*D_height*D_width]);
D = permute(D,[2 1]); %[Num_D*D_height*D_width Channel_D]

W = permute(W,[4 3 2 1]); %[Filter_width Filter_heigt Channel_Filter Num_Filter]
W = reshape(W,[Channel_Filter*Filter_height*Filter_width Num_Filter]);
W = permute(W,[2 1]); %[Num_Filter Channel_Filter*Filter_heigt*Filter_width]

col_D = D*W; %[Num_D*D_height*D_width Channel_Filter*Filter_heigt*Filter_width]
dcx = Col2Im(col_D,Output_shape,[Filter_height,Filter_width],Stride,Padding);
dcx = (Output_shape>0).*dcx; %ReLU の逆伝播
end
```

Updata_Parameters_Momentum3

```
function
[Filters,Weights,bias_c,bias_t,Moment_c,biasMoment_c,Moment_t,biasMoment_t]=Update_P
arameters_Momentum3(D_t, ...
```

```
    D_c,X_t,X_c,X0,Filters, Weights,bias_c,bias_t,Eta,
Beta,Moment_t,Moment_c,biasMoment_t,biasMoment_c,Setting)

L1 = Setting{1};
L2 = Setting{2};

for k = L2:-1:2
    % Weights の更新
    Grad_weight = D_t{1,k}'*X_t{1,k-1};
    Moment_t{1,k} = Eta*Grad_weight - Beta*Moment_t{1,k}; % 左辺を Moment と称してい
るが，この名称はあまり良くない．以下同様．
    Weights{1,k} = Weights{1,k} - Moment_t{1,k};
    % bias_t の更新
    Grad_bias_t = sum(D_t{1,k},1);
    biasMoment_t{1,k} = Eta*Grad_bias_t - Beta*biasMoment_t{1,k};
    bias_t{1,k} = bias_t{1,k} -biasMoment_t{1,k};
end

% Weights の更新
Moment_t{1,1} = Eta*(D_t{1,1}'*X_c{1,2}) - Beta*Moment_t{1,1};
Weights{1,1} = Weights{1,1} - Moment_t{1,1};
% bias_t の更新
D_bias = sum(D_t{1,1},1);
biasMoment_t{1,1} = Eta*D_bias - Beta*biasMoment_t{1,1};
bias_t{1,1} = bias_t{1,1} - biasMoment_t{1,1};

for k = L1:-1:2
    % Filter の更新
    Moment_c{1,k} = Eta*Grad_filter(X_c{1,k-1},D_c{1,k},Filters{1,k})-
Beta*Moment_c{1,k} ;
    Filters{1,k} = Filters{1,k} - Moment_c{1,k};
    % bias_c の更新
    biasMoment_c{1,k} = Eta*Grad_bias_c(D_c{1,k})-Beta*biasMoment_c{1,k};
    bias_c{1,k} = bias_c{1,k} -biasMoment_c{1,k};
end

% Filter の更新
Moment_c{1,1} = Eta*Grad_filter(X0,D_c{1,1},Filters{1,1})-Beta*Moment_c{1,1};
Filters{1,1} = Filters{1,1} - Moment_c{1,1};
% bias の更新
biasMoment_c{1,1} = Eta*Grad_bias_c(D_c{1,1})-Beta*biasMoment_c{1,1};
bias_c{1,1} = bias_c{1,1} -biasMoment_c{1,1};

end
```

Grad_filter

```
function COL = Grad_filter(X,D,F)

% 下記参考文献 [3] の Python プログラムに基づく．
```

```
[NumFilter,ChannelFilter,Filter_height,Filter_width] = size(F);
[NumDelta,ChannelDelta,Delta_height,Delta_width] = size(D);

D = permute(D,[4 3 1 2]);
D = reshape(D,[NumDelta*Delta_height*Delta_width ChannelDelta]);
D = permute(D,[2 1]);

s=fix(size(F,3)/2);
Col_X = Im2Col(X,[Filter_height,Filter_width],1,s);
COL = D*Col_X;

COL = permute(COL,[2 1]);
COL = reshape(COL,[Filter_width Filter_height ChannelFilter NumFilter]);
COL = permute(COL, [4 3 2 1]);
end
```

Grad_bias_c

```
function B = Grad_bias_c(D)

% 下記参考文献 [3] の Python プログラムに基づく.
[NumDelta,ChannelDelta,Delta_height,Delta_width] = size(D);
D = permute(D,[4 3 1 2]);
D = reshape(D,[NumDelta*Delta_height*Delta_width ChannelDelta]);
D = permute(D,[2 1]);
B = sum(D,2)';
end
```

Im2Col

```
function Col = Im2Col(Image,block,Stride,Padding)

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 入力タイプ
% Image = Image(Number of Filter, Channel, Image_height, Image_width)
% block = [number1, number2]
% Stride = number, Padding = number
%
% 出力 Col のサイズ :
% (NumImage*Output_height * Output_width) x (Channel * Block_height*Block_width)
% ここで
% Output_hight = fix((Image_height-Block_height+2*Padding)/Stride)+1
% Output_width = fix((Image_width-Block_width+2*Padding)/Stride)+1
%
% 参考文献 [1], [2], [3], [4] の Python プログラムにもとづく.
%%%%%%%%%%%%%%%%%%%%%%%%%%%

Block_height = block(1);
Block_width = block(2);
[NumImage, Channel, Image_height, Image_width] = size(Image);
```

```
Output_hight = fix((Image_height-Block_height+2*Padding)/Stride)+1;
Output_width = fix((Image_width-Block_width+2*Padding)/Stride)+1;
col = zeros(NumImage,Block_height*Block_width,Channel,Output_hight,Output_width);

Image = padarray(Image,[0 0 Padding Padding],0,'both');

HS=Output_hight*Stride;
WS=Output_width*Stride;

for h=1:Block_height
    for w = 1:Block_width
        % MATLAB の im2col の配列にするには次のようにする.
        col(:,
(h-1)*Block_width+w,:,:,:)=Image(:,:,w:Stride:w-1+WS ,h:Stride:h-1+HS);
        % この部分は，Python などの他の文献のようにするには次のようにする.
        % col(:,(h-1)*Block_width+w,:,:,:)=Image(:,:, h:Stride:h-1+HS,
w:Stride:w-1+WS);
    end
end

% MATLAB 式 im2col にするには次のようにする：
 Col = permute(col,[2 3 4 5 1]);
% この部分は，Python などの他の文献のようにするには次のようにする.
% Col = permute(col,[2 3 5 4 1]);

Col = reshape(Col, [Channel*Block_height*Block_width
NumImage*Output_hight*Output_width ]);
Col = permute(Col,[2 1]);
end
```

Col2Im

```
function Img = Col2Im(Col, Image_type, Block_size,Stride,Padding)

%%%%%%%%%%%%%%%%%%%%%%%%%%
% 入力と出力のタイプ - 考え方：
% Col = Im2Col(Image,block,Stride,Padding) に対して，
% Img = Col2Im(Col,Image_type,block,Stride,Padding) となっていることを想定している.
% したがって
% [NumImage, Channel, Image_height, Image_width] = size(Image)
% に対して，Col2Im に入力する
%
% Image_type は [NumImage, Channel, Image_height, Image_width]
%
% である.
%
% block はデータを区切りブロック化するサイズを指定．畳み込みのフィルタのサイズに相当.
% Stride はデータをずらす幅．畳み込みのずらし幅に相当.
% Padding は画像のゼロパディングの幅．畳み込みの出力サイズを定める幅に相当.
%
```

```
% Col2Im の出力サイズは
% [NumFilter,Channel,Image_height,Image_width] = size(Image_type);
%
% [2], [3], [4] の Python プログラムにもとづく.
%%%%%%%%%%%%%%%%%%%%%%%%%%

Block_height = Block_size(1);
Block_width  = Block_size(2);
[NumFilter,Channel,Image_height,Image_width] = size(Image_type);
Output_height = fix((Image_height - Block_height + 2*Padding)/Stride)+1;
Output_width = fix((Image_width - Block_width + 2*Padding)/Stride)+1;

Col = permute(Col, [2 1]);
Col = reshape(Col,[Block_height*Block_width Channel Output_width Output_height
NumFilter]);
Col = permute(Col,[5 1 2 4 3]); %[NumFilter Block_height*Block_width Channel
Output_height Output_width]

im_scheme = zeros(NumFilter,Channel,Image_height+2*Padding+Stride-1, ...
    Image_width+2*Padding+Stride-1);
for h=1:Block_height
    for w=1:Block_width
        coll = zeros(NumFilter,Channel,Output_height,Output_width);
        coll(:,:,:,:) = Col(:,(h-1)*Block_width+w,:,:,:);
        im_scheme(:,:, h:Stride:(h-1)+Output_height*Stride, w:Stride:
(w-1)+Output_width*Stride) = im_scheme(:,:, ...
            h:Stride:(h-1)+Output_height*Stride, w:Stride:(w-1)+Output_width*Stride)
+coll(:,:,:,:);
    end
end
Img = im_scheme(:,:,Padding+1:Image_height+Padding,Padding+1:Image_width+Padding);
end
```

プログラムの参考文献

[1] https://docs.chainer.org/en/v7.8.1.post1/reference/generated/chainer.functions.im2col.html

[2] 斎藤康毅，ゼロから作る Deep Learning，O'REILLY, 2016.

[3] 立石賢吾，やさしく学ぶディープラーニングがわかる数学のきほん，マイナビ，2019.

[4] 我妻幸長，はじめてのディープラーニング - Python で学ぶニューラルネットワークとバックプロパゲーション，SB Creative, 2018

数学者が書いた深層学習講義,　畳み込みニューラルネット篇 MATLAB プログラム付き
著者 新井仁之
第 1 版 Ver.1.1 2025 年 1 月 6 日
第 1 版 Ver.1.2 2025 年 1 月 7 日
第 1 版 Ver.1.3 2025 年 1 月 8 日
第 1 版 Ver.1.4 2025 年 1 月 10 日 Appendix Ver.1.3 2025 年 1 月 12 日
第 1 版 Ver.1.5 2025 年 1 月 14 日
発行 数学と錯視の科学館 電子出版部

発行　数学と錯視の科学館 電子出版部