電気電子プログラミング及演習 総合課題「手書き文字認識」

村脇 有吾　櫻田 健

2024/4/1

総合課題は，手書きの数字 0〜9 をニューラルネット（以下 NN）によって認識するC言語プログラムの作成である．

1　NN による手書き文字認識

まず，NN による手書き文字認識の基本的な考え方を説明する．

入力として与えられるのは，数字 0〜9 のいずれか 1 文字を手書きした 28×28 画素のグレイスケール画像（モノクロ画像）である（図 1 左）．各画素は $[0:1]$ の実数値で濃淡を表す（0 が黒，1 が白）．

２次元状に配置された画素群は，計算機では１次元配列で表現される．例えば $(6,3)$ の画素は，配列中では添字 $3 \times 28 + 6 = 90$ の画素となる（図 1 右）．すなわち，28×28 の画像は各要素が $[0:1]$ の実数値をとる $28 \times 28 = 784$ 次元の列ベクトルとなる[*1]．以降，この 784 次元の入力ベクトルを $\boldsymbol{x}$ とする．

出力は，「文字 0〜9 のいずれか」を各文字の確率で表現する 10 次元ベクトル $\boldsymbol{y}$ である．たとえば，$\boldsymbol{y} = (0,0,1,0,0,0,0,0,0,0)^\top$ は文字 2 以外ありえないと判定したことを意味し，$\boldsymbol{y} = (0.1,0,0,0,0,0.3,0,0.6,0,0)^\top$ は文字 7 の可能性が最も高く，次いで文字 5，0 の可能性があることを示す．

NN による文字認識では，入力ベクトル $\boldsymbol{x}$ に対して出力ベクトル $\boldsymbol{y}$ を計算する関数 $\boldsymbol{y} = f(\boldsymbol{x})$ を用意する．この段階を**学習**（training）とよぶ．その上で，与えられた文字画像 $\boldsymbol{x}$ に対して $\boldsymbol{y} = f(\boldsymbol{x})$ を計算し，$\boldsymbol{y}$ の中の最大要素に対応する数字を認識結果とする．この段階を**推論**（inference）とよぶ．

1.1　NN の構成

前述の関数 $f(\boldsymbol{x})$ を NN で構成する方法は無数に考えられるが，今回の総合課題では図 2 の 6 層 NN を用いる．各層は以下の 3 種類の層からなる．以下の説明では各層の入力を $\boldsymbol{x}$，出力を $\boldsymbol{y}$ とし，それぞれの k 番目の要素を x_k, y_k とする[*2]．

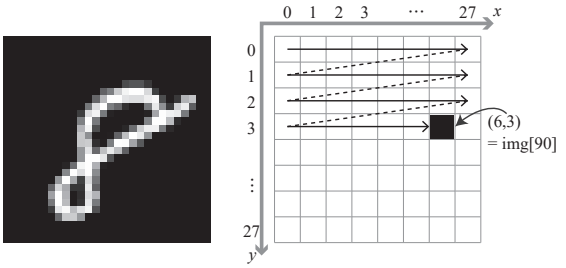

図 1　手書き文字画像．左：画像例．右：画像座標系

■全結合層　全結合層（fully-connected layer, FC 層）は，n 次元ベクトルの入力 $\boldsymbol{x}$ に対して，$m \times n$ 行列 A と m 次元ベクトル $\boldsymbol{b}$ によって $A\boldsymbol{x} + \boldsymbol{b}$ を出力する．行列 A の第 k 行ベクトルを $\boldsymbol{a}_k$，ベクトル $\boldsymbol{b}$ の第 k 要素を b_k とすると，

$$y_k = \boldsymbol{a}_k \boldsymbol{x} + b_k \ (k = 1, \ldots, m) \tag{1}$$

となる．行列 A のサイズ $n \times m$ は FC1, FC2, FC3 についてそれぞれ $50 \times 768, 100 \times 50, 10 \times 100$ とする．行列 A およびベクトル $\boldsymbol{b}$ の具体的な値は後述の学習によって決定される．

■活性化層　活性化層では ReLU (Rectified Linear Unit) 関数による以下の計算を行う．

$$y_k = \begin{cases} x_k & (x_k > 0) \\ 0 & (otherwise) \end{cases} \tag{2}$$

入力ベクトルと出力ベクトルの次元数は変化せず，FC1, FC2, FC3 との接続関係から ReLU1, ReLU2 はそれぞれ 50 次元，100 次元となる．

■出力層　出力層はソフトマックス関数

$$y_k = \frac{\exp(x_k)}{\sum_i \exp(x_i)} \tag{3}$$

[*1] 以降特に断りがない場合，ベクトルとは列ベクトルを意味する．

[*2] 先に NN 全体の入出力も $\boldsymbol{x}$，$\boldsymbol{y}$ としたので，層の説明との区別に注意すること．

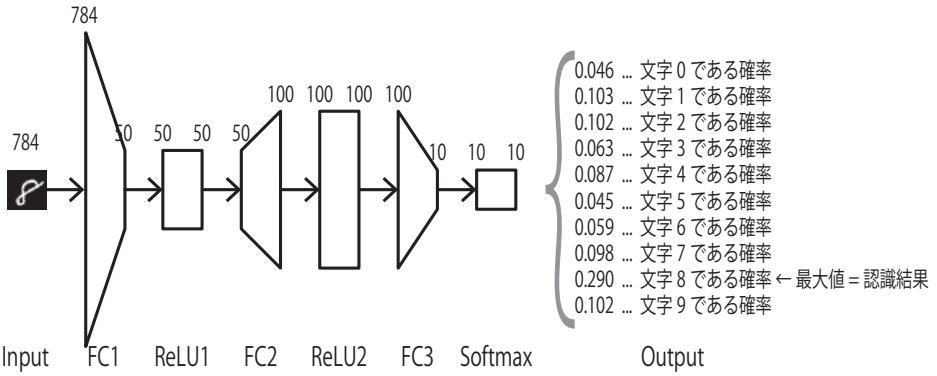

図 2　ニューラルネットワークの構成

によって，入力ベクトル $\boldsymbol{x}$ の各要素 x_k に対して e^{x_k} を計算して正規化したものを出力ベクトル $\boldsymbol{y}$ とする（次元数はともに 10）．$\boldsymbol{y}$ は要素の和が 1 となり，各要素の値を確率として扱うことができる．

実装上は，x_k の最大値 $x_{\max}$ を用いて

$$y_k = \frac{\exp(x_k - x_{\max})}{\sum_i \exp(x_i - x_{\max})} \tag{4}$$

とする（式変形すれば等価であることがわかる）．これは，正の数で exp を計算すると，容易に大きな値となりオーバーフローする（計算機で表現できる数値の範囲を超える）ためである．

1.2　NN による推論と誤差

NN による文字認識は

$$\boldsymbol{y} = s(f_3(a_2(f_2(a_1(f_1(\boldsymbol{x})))))) \tag{5}$$

を計算し，得られたベクトル $\boldsymbol{y}$ の中の最大値となる要素番号を判別結果とすることに相当する．ただし関数 $f_i(\cdot)$ は i 番目の全結合層 FCi における式 (1) を，関数 $a_i(\cdot)$ は i 番目の活性化層 ReLUi における式 (2) を，関数 $s(\cdot)$ は出力層における式 (4) をそれぞれ表す．

NN による認識の正しさは，入力 $\boldsymbol{x}$ から NN によって計算された出力 $\boldsymbol{y}$ と，入力 $\boldsymbol{x}$ に対して（人手によって作成された）本来出力されるべき正解 $\boldsymbol{t}$ を比較することで計算できる．例えば $\boldsymbol{x}$ が数字 3 を表す画像のとき $\boldsymbol{t} = (0,0,0,1,0,0,0,0,0,0)^\top$ と表現する（これを one-hot 表現とよぶ）．このとき，$\boldsymbol{y}$ と $\boldsymbol{t}$ の誤差（loss）を，2 つの確率分布間の尺度である交差エントロピー（cross entropy）を用いて次のように定義する[*3]．これを損失関数（loss function）とよぶ．

$$E = -\sum_k t_k \log y_k \tag{6}$$

この値が小さいほど NN の認識結果が正解に近いといえるので，次節以降で述べる NN の学習はこの値を減らすことを目標として進められる．

1.3　NN の学習

1.3.1　勾配法

前述のように NN の良さは交差エントロピー誤差 E によって定義され，その大小は NN を構成する各層のパラメータによって決まる．今回の NN は各全結合層 FCi のみがパラメータ $A_i, \boldsymbol{b}_i$ $(i = 1,2,3)$ を持ち，その総数は $(784 \times 50 + 50) + (50 \times 100 + 100) + (100 \times 10 + 10) = 45460$ である．すなわち，交差エントロピー誤差はこの 45460 次元ベクトル $\boldsymbol{\theta}$ を変数とする非線形スカラ関数 $E(\boldsymbol{\theta})$ とみなすことができる．

このとき，損失関数 $E(\boldsymbol{\theta})$ を最小化する $\boldsymbol{\theta}$ は勾配法によって推定することができる．勾配法とは，まず乱数などによって $\boldsymbol{\theta}$ の初期値 $\boldsymbol{\theta}^{(0)}$ を与え，$\boldsymbol{\theta}^{(0)}$ から $E(\boldsymbol{\theta}^{(0)}) > E(\boldsymbol{\theta}^{(1)}) > E(\boldsymbol{\theta}^{(2)}), \ldots$ となるように $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ と徐々にパラメータの値を更新することにより最適な $\boldsymbol{\theta}$ を推定するアルゴリズムである．

具体的には変数 $\boldsymbol{\theta}$ の各要素 θ_k に対して，損失関数 $E(\boldsymbol{\theta})$ の偏微分 $\frac{\partial E}{\partial \theta_k}$ を計算する．

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \left(\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots, \frac{\partial E}{\partial \theta_k}, \ldots, \frac{\partial E}{\partial \theta_{45460}} \right)^\top \tag{7}$$

このベクトル（勾配（gradient）とよぶ）が点 $\boldsymbol{\theta}$ において $E(\boldsymbol{\theta})$ が最も増大する方向を示していることを

[*3] 交差エントロピー誤差を用いるのは，NN の学習時の計算を効率的に行うためである．次節の式 11 参照．

利用して，現在の推定値 $\boldsymbol{\theta}^{(t)}$ から次の推定値 $\boldsymbol{\theta}^{(t+1)}$ を

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}^{(t)}} \tag{8}$$

によって計算する．ここで係数 η はどれだけ勾配方向に移動するかを決めるハイパーパラメータであり，学習率（learning rate）とよばれる．

1.3.2 誤差逆伝播法

勾配法によって最適な $\boldsymbol{\theta}$ を得るためには，勾配 $\frac{\partial E}{\partial \boldsymbol{\theta}}$ を繰り返し計算する必要がある．一般に計算機によって勾配を求める方法は**数値的な（numerical）微分**と**解析的な（analytic）微分**の 2 通りがある．前者は微分の定義に従って微小量 ϵ および k 番目の要素が ϵ でそれ以外が 0 となるベクトル $\boldsymbol{\epsilon}_k = (\dots, 0, \epsilon, 0, \dots)^\top$ を用いた中心差分

$$\frac{\partial E}{\partial \theta_k} \approx \frac{E(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k) - E(\boldsymbol{\theta} - \boldsymbol{\epsilon}_k)}{2\epsilon} \tag{9}$$

によって各偏微分の値を数値的に求める．この方法は式 (6) で定義された損失関数 E の計算さえできれば実現できるため実装が容易であるが，勾配を 1 回計算するために損失関数を要素の数だけ（今回の例では 45460 要素×「$\pm\epsilon$」2 回＝合計 90920 回）計算する必要があり，実行速度の点で実用的ではない．

一方，後者の解析的な方法とは，代数的に偏微分の式を導出しておき，これに現在の $\boldsymbol{\theta}$ を代入することで勾配の値を直接計算する方法である．前者の数値微分と比較すると明らかに計算量が少ないが，NN の構成を変更するたびに偏微分の式を導出してプログラムに反映させなくてはならない点に問題がある．この問題を解決する方法が誤差逆伝播法（backward propagation）である．

誤差逆伝播法のポイントは，損失関数 E が簡単な要素関数（FC, ReLU, Softmax）の合成関数になっていることに着目し，微分の連鎖律（chain rule）に基づいて全体の偏微分を各要素関数の偏微分の積に分解して計算する点にある[*4]．これによって，各要素関数の偏微分さえ事前に導出しておけば，層の構成を変更しても連鎖律に沿って偏微分を直接計算することができる．

具体的には，各層において上流側（出力側）から勾配 $\frac{\partial E}{\partial \boldsymbol{y}}$ が与えられたとき，その層のパラメータの勾配と，下流側（入力側）に与える勾配 $\frac{\partial E}{\partial \boldsymbol{x}}$ を計算し，より下流側へと逆順に計算を進めていく．今回考える NN の各層では次のように計算を行う．

■損失関数＋ソフトマックス層　損失関数とソフトマックス層を組み合わせると，式 (3) と式 (6) から

$$\begin{aligned} E &= -\sum_k t_k \log y_k \\ &= -\sum_k t_k \log \frac{\exp(x_k)}{\sum_{i=1}^n \exp(x_i)} \end{aligned} \tag{10}$$

となり，勾配は次のよう計算される．

$$\frac{\partial E}{\partial x_k} = y_k - t_k \tag{11}$$

（各自この偏微分を確認すること．）

■ReLU 層　式 (2) より

$$\frac{dy_k}{dx_k} = \begin{cases} 1 & (x_k > 0) \\ 0 & (otherwise) \end{cases} \tag{12}$$

となる．したがって上流側の勾配 $\frac{\partial E}{\partial y_k}$ および現在の x_k が与えられたとき，

$$\frac{\partial E}{\partial x_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial x_k} = \begin{cases} \frac{\partial E}{\partial y_k} & (x_k > 0) \\ 0 & (otherwise) \end{cases} \tag{13}$$

すなわち，x_k が正なら上流の値をそのまま下流に転送し，それ以外は 0 とする．

■全結合層　式 (1) より，パラメータ A,$\boldsymbol{b}$ を更新するための勾配は次のように計算できる（$\boldsymbol{a}_k$ は A の第 k 行ベクトル）．

$$\frac{\partial E}{\partial \boldsymbol{a}_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial \boldsymbol{a}_k} = \frac{\partial E}{\partial y_k}\boldsymbol{x}^\top \tag{14}$$

$$\frac{\partial E}{\partial b_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial b_k} = \frac{\partial E}{\partial y_k} \tag{15}$$

さらに，下流へ転送する勾配は次のようになる．

$$\frac{\partial E}{\partial x_k} = \sum_j \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_k} = \boldsymbol{a}_k^\top \frac{\partial E}{\partial \boldsymbol{y}} \tag{16}$$

ただし，ここでの $\boldsymbol{a}_k$ は A の第 k 列ベクトルである[*5]．

以上により誤差逆伝播法によるパラメータの更新は以下のように実装できる．

[*4] 例えば，$z = t^2$，$t = x + y$ のとき，$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$ となる．

[*5] 式 (16) を具体例で各自確認すること．例えば，A を 3×2 とすると $y = A\boldsymbol{x} + \boldsymbol{b}$ は次のようになる．

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + b_1 \\ a_{21}x_1 + a_{22}x_2 + b_2 \\ a_{31}x_1 + a_{32}x_2 + b_3 \end{pmatrix}$$

Step 1. ある $\boldsymbol{x}$ に対して順伝播により $\boldsymbol{y}$ を計算する．このとき FC および ReLU 層では式 (14) および式 (13) のためにその層の入力 $\boldsymbol{x}$ を記憶しておく．

Step 2. 得られた $\boldsymbol{y}$ と，$\boldsymbol{x}$ に対応する正解ベクトル $\boldsymbol{t}$ を用いて，式 (11) により最終層から FC3 に向かう勾配を計算する．

Step 3. FC3 において式 (14),(15),(16) により A_3, $\boldsymbol{b}_3$，および ReLU2 に向かう勾配を計算する（式 (13) で計算された $\frac{\partial E}{\partial \boldsymbol{x}}$ をここでの $\frac{\partial E}{\partial \boldsymbol{y}}$ とする）．

Step 4. ReLU2 において式 (13) により FC2 に向かう勾配を計算する．

Step 5. FC2 において式 (14),(15),(16) により A_2, $\boldsymbol{b}_2$, および ReLU1 に向かう勾配を計算する．

Step 6. ReLU1 において式 (13) により FC1 に向かう勾配を計算する．

Step 7. FC1 において式 (14),(15) により A_1 と $\boldsymbol{b}_1$ の勾配を計算する．

Step 8. 式 (8) に基づき，$A_i \leftarrow A_i - \eta \frac{\partial E}{\partial A_i}$ および $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - \eta \frac{\partial E}{\partial \boldsymbol{b}_i}$ によってパラメータを更新する．

1.4 確率的勾配降下法（SGD）

正解が既知の手書き文字画像 N 枚（具体的に $N = 60,000$ とする）を訓練データとして NN を学習することを考える．このとき誤差逆伝播法によってパラメータ $\boldsymbol{\theta}$ を更新する方法は以下のようにいろいろと考えられる．

方法 1 画像を 1 枚選んで勾配を計算し，パラメータを更新する．

方法 2 画像を 1000 枚選んで勾配を計算し，その平均を使ってパラメータを更新する．

方法 3 画像 60000 枚すべての勾配を計算し，その平均を使ってパラメータを更新する．

一般に用いられる方法は**確率的勾配降下法**（Stochastic Gradient Descent, SGD）とよばれる方法で，

> 画像を n 枚（$n \ll N$）ランダムに選び，各画像について勾配を計算してその平均を使ってパラメータを更新する．

という操作を繰り返す方法である．このとき選ばれた n 枚の画像集合のことをミニバッチとよび，このような小さな単位で学習を繰り返すことをミニバッチ学習とよぶ．仮に $n = 100$ とすると 600 回この操作を繰り返すことで訓練データすべてを試したことになる（NN がすべてのデータを見た，と表現する）．これをエポックとよび，この例ではミニバッチ学習 600 回が 1 エポックに相当する．

1.4.1 過学習

SGD によって学習を行うと訓練データに対して損失関数が最小となるパラメータが学習されるが，訓練データに対して過度に特化した最適化が行われ，他の画像を正しく認識できない過学習状態に陥る可能性がある．そこで 1 エポックなど何らかの区切りごとに，訓練データとは別のテストデータに対して訓練データと同等の精度で認識ができるかどうかを確認する．もし訓練データに対する精度とテストデータに対する精度に大きな差が生じた場合は過学習である．

1.5 全体のアルゴリズム

以上をまとめると，NN による文字認識のアルゴリズムの全体は以下のようになる．

■学習

Step 1. それぞれに正解が与えられている訓練データとテストデータを用意する．また，学習率 η およびミニバッチサイズ n を決める．

Step 2. 確率的勾配降下法によりミニバッチ学習を１エポック行う．

Step 3. 訓練データとテストデータに対する認識精度を比較する．大きな差がある場合は失敗とみなして学習を打ち切る．

Step 4. 認識精度が所望の値に達するか，規定の回数だけ学習を繰り返したら，その時点のパラメータを学習結果として保存し終了する．そうでなければ Step. 2 に戻る．

■推論（文字認識） 学習によって得られた値を NN のパラメータとして読み込み，入力画像 $\boldsymbol{x}$ に対して出力 $\boldsymbol{y}$ を計算する．得られた $\boldsymbol{y}$ のなかで最大値となる要素に対応する文字を認識結果とする．

2 サンプルコード，データ

今回の総合課題では Yann LeCun が公開している MNIST データ [1] を使用する．訓練・テストデータおよびサンプルコードが PandA の総合課題の項目にあるのでダウンロードして利用すること．

data-samplecode.zip には以下のファイルが含まれている．

train-images-idx3-ubyte MNIST 訓練用画像データ．大きさ 28×28，60000 枚

train-labels-idx1-ubyte MNIST 訓練用ラベルデータ．

t10k-images-idx3-ubyte MNIST テスト用画像データ．大きさ 28×28，10000 枚

t10k-labels-idx1-ubyte MNIST テスト用ラベルデータ．

nn.h 補助関数が定義されたヘッダファイル

example.c ソースコードのテンプレート

example.c には，訓練データとテストデータの読み込み方法，デバッグ用に特定の画像データを BMP 画像としてファイル出力する方法が示されている．

3 総合課題

これまで説明したC言語による手書文字認識プログラムを作成すること．口頭試問においてレポートを提出し，プログラムのデモを行い，試問に答えること．

3.1 プログラムの必要要件

以下の要件を満たすプログラムを作成すること．なお，学習動作と推論動作は1つのプログラムのオプションによる切り替えでも，別々のプログラムでもよい．

■学習動作について

- □ 乱数によってパラメータを初期化し，MNIST の 60000 画像を訓練データとして SGD により学習を行う．
- □ 学習1エポックごとに訓練データとテストデータに対する損失関数の値と認識精度を表示する．
- □ 学習されたパラメータを実行時引数で指定されたファイルに保存する．

■推論動作について

- □ 実行時に引数で指定された (1)28×28 の BMP ファイルと，(2) 自らの学習プログラムが保存したパラメータファイルを読み込み，認識結果を画面出力する．

■プログラムの記述について

- □ 各ブロック（{から対応する}まで）に適切なインデントがつけられていること．
- □ オプション-Wall を有効にしてコンパイルしたときに，警告が1つも出ないこと．
- □ 共通化できる処理を関数にまとめるなど，構造化されたプログラムとすること（安易にコピー&ペーストを繰り返して冗長なプログラムにしない）．1つの関数は概ね 50 行以内とすること．
- □ プログラム中の関数名，変数名には意味が明確な名称を与えること．
- □ 関数および重要な変数にはコメントを付けること．少なくとも作成した各関数の引数と戻り値の意味をコメントとして記述すること．

3.2 工夫・拡張・考察等について

必須要件に加えて，プログラム作成において工夫・拡張を行ったり，NN の学習について考察を行った場合にはレポートに記載すること．以下は一例であるが，これらに限らず自由な発想に基づく試みを積極的に評価する．

- □ 訓練データと認識精度の関係について考察する．訓練データ量を 1/2,1/4,1/8 などにしてみる，クラスの偏り（いくつかの数字のデータを減らす）など．
- □ バッチサイズを変化させた場合，また，バッチのデータをランダムに選ぶのではなく，クラスの偏りをもたせた場合（正解の数字でソートするなど）の学習の振舞いを考察する．
- □ NN の構成を変化させた場合（中間層の数，各層の次元数など）の認識精度について考察する．
- □ パラメータの初期値を変えて学習の振舞いを考察する．すべて 0 にする，一様乱数にする，ガウス分布にするなど（入力が n 次元のとき，平均 0，分散 $\sqrt{2/n}$ のガウス分布を用いる方法がある [2]）．

3.3 レポートの必須要件

レポートでは以下の点を判りやすく，少なくとも3つの節に分けて説明すること．

- □ **プログラムの概要**：入力引数，処理の流れなどを説明すること．
- □ **関数の説明**：作成した各関数の引数・戻り値の意味および期待される値の範囲（例えば非負である，など），さらに関数が行う処理の概要を説明すること．
- □ **工夫・拡張・考察事項**：プログラムで工夫・拡張した点，NN の学習について考察した事項等を説明すること．

総合課題提出に先立ち，以上の各項目について確認して□欄にチェックをいれて確認することを推奨する．

3.4 評価方法

総合課題の評価は，プログラム動作・記述に関する必須要件およびレポートについての必須要件をすべて満たし，試問で適切なデモおよび回答をすることによって「A〜B」相当とする．これらに加え，プログラムの工夫・拡張・考察等を行ったものを「A+〜A」相当とする．

講義全体の評価・合否は，毎週の課題の評価と総合課題の評価を総合して決定する．総合課題を解いて口頭試問に参加することは単位取得の必要条件である．

参考文献

[1] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`. Last accessed: March 31, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[3] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. CS231n: Convolutional neural networks for visual recognition. `http://cs231n.github.io/`. Last accessed: March 31, 2017.

[4] 斎藤 康毅. **ゼロから作る** *Deep Learning* **–** *Python* **で学ぶディープラーニングの理論と実装**. O'Reilly Japan, 2016.

付録 A　補題

以下は総合課題に必要となる関数を実装する補題である．参考までに，補題 15 の 3 層 NN まで実装して合計 200 行程度，総合課題の必須要件まで実装して合計 300 行程度のコード量である．

補題 1：行列の表示

float 型の 1 次元配列として与えられた $m \times n$ 行列 x が入力されたときに，その内容を表示する関数として図 3 の仕様を満たすものを作成せよ．

補題 2：行列の積和演算

float 型の 1 次元配列として与えられた $m \times n$ 行列 A，float 型の 1 次元配列として与えられた m 行の列ベクトル b，float 型の 1 次元配列として与えられた n 行の列ベクトル x が入力されたときに式 (1) を計算し，その結果を float 型の 1 次元配列として与えられた m 行の列ベクトル y に書き込む関数として図 4 の仕様を満たすものを作成せよ．

補題 3：ReLU 演算

float 型の 1 次元配列として与えられた n 行の列ベクトル x が入力されたときに，式 (2) を計算し，float 型の 1 次元配列として与えられた n 行の列ベクトル y に書き込む関数として図 5 の仕様を満たすものを作成せよ．

補題 4：Softmax 演算

float 型の 1 次元配列として与えられた n 行の列ベクトル x が入力されたときに，式 (4) を計算し，float 型の 1 次元配列として与えられた n 行の列ベクトル y に書き込む関数として図 6 の仕様を満たすものを作成せよ．

補題 5：推論（3 層）

float 型の 1 次元配列として与えられた 10×784 行の行列 A，float 型の 1 次元配列として与えられた 10 行の列ベクトル b，float 型の 1 次元配列として与えられた 784 行の列ベクトル x が入力されたときに，式 (1), (2), (4) を順に計算し，出力 $\boldsymbol{y}$ の各要素のうち最大となるものの添字 $[0:9]$ を返す関数として図 7 の仕様を満たすものを作成せよ．正しく実装できている場合，推論結果 ans および正解値 train_y[0] はともに 5 となるはずである．

補題 6：画像保存

train_x, test_x を実際にファイル出力するための関数として save_mnist_bmp() が用意されている．図 8 に従い，実際に train_x の 0 番目画像を保存し，得られたファイルを適当なアプリケーションで画面表示して数字を確認せよ．

補題 7：正解率（3 層）

図 9 に示すように, 上記の 3 層 NN によって test_x〜test_x+(test_count-1)*784 を入力して推論を行い，正解 test_y[0]〜test_y[test_count-1] と比較することで正解率を計算せよ．正しく実装できている場合，正解率は 92.31% となるはずである．

補題 8：誤差逆伝播（Softmax 層）

式 (11) により $\frac{\partial E}{\partial \boldsymbol{x}}$ を計算する関数として図 10 の仕様を満たすものを作成せよ．ただし $\frac{\partial E}{\partial \boldsymbol{x}}$ に対応する引数 dEdx は引数 y と同じ大きさの配列へのポインタであるとする．また引数 t は正解を表す $[0:9]$ の整数である．

補題 9：誤差逆伝播（ReLU 層）

式 (13) によって $\frac{\partial E}{\partial \boldsymbol{x}}$ を計算する関数として図 11 の仕様を満たすものを作成せよ．ただし $\frac{\partial E}{\partial \boldsymbol{x}}$ に対応する引数 dEdx は引数 x と同じ大きさの配列へのポインタであるとする．

補題 10：誤差逆伝播（FC 層）

式 (14),(15), (16) によって $\frac{\partial E}{\partial A}$, $\frac{\partial E}{\partial \boldsymbol{b}}$, $\frac{\partial E}{\partial \boldsymbol{x}}$ を計算する関数として図 12 の仕様を満たすものを作成せよ．ただし $\frac{\partial E}{\partial A}$ に対応する引数 dEdA は引数 A と，$\frac{\partial E}{\partial \boldsymbol{b}}$ に対応する引数 dEdb は引数 b と，$\frac{\partial E}{\partial \boldsymbol{x}}$ に対応する引数 dEdx は引数 x と同じ大きさの配列へのポインタであるとする．

補題 11：誤差逆伝播（3 層）

第 1.3 節と同様に 3 層の場合の誤差逆伝播を行う関数として図 13 の仕様を満たすものを作成せよ．

Step 1. 適当な入力として train_x+784*8 を $\boldsymbol{x}$ として選んで推論（3 層）によって $\boldsymbol{y}$ を計算する．このとき FC および ReLU 層では式 (16) および式 (13) のためにその層の入力を記憶しておく．

Step 2. 得られた $\boldsymbol{y}$ と, この $\boldsymbol{x}$ に対応する正解ベクトル $\boldsymbol{t}$（今回の例では train_y[8]）を用いて，式 (11) により最終層から ReLU に向かう勾配を計算する（関数 softmaxwithloss_bwd を使う）．

Step 3. ReLU において式 (13) により FC に向かう勾配を計算する（関数 relu_bwd の引数 dEdy に，関数 softmaxwithloss_bwd で計

```
#include "nn.h"
void print(int m, int n, const float * x) {
  // 自分で実装
}
// テスト
int main() {
  print(1, 10, b_784x10);
  return 0;
}
```

出力

```
-0.4187  0.3734  0.1757 -0.2988  0.0419  1.6848 -0.0324  0.9227 -1.6611 -0.2257
```

図 3　print() と実行例

```
#include "nn.h"
void fc(int m, int n, const float * x, const float * A, const float * b, float * y) {
  // 自分で実装
}
// テスト
int main() {
 float * train_x = NULL;
 unsigned char * train_y = NULL;
 int train_count = -1;
 float * test_x = NULL;
 unsigned char * test_y = NULL;
 int test_count = -1;
 int width = -1;
 int height = -1;
 load_mnist(&train_x, &train_y, &train_count,
            &test_x, &test_y, &test_count,
            &width, &height);
 // これ以降，3層 NN の係数 A_784x10 および b_784x10 と，
 // 訓練データ train_x + 784*i (i=0,...,train_count-1), train_y[0]〜train_y[train_count-1],
 // テストデータ test_x + 784*i (i=0,...,test_count-1), test_y[0]〜test_y[test_count-1],
 // を使用することができる．
 float y[10];
 fc(10, 784, train_x, A_784x10, b_784x10, y);
 print(1, 10, y);
 return 0;
}
```

出力

```
  1.6129 -4.6008  2.0135  6.3752 -5.8886  8.4595 -1.1602  1.4323  1.2656  0.3464
```

図 4　fc()

算した dEdx を入れる）．

Step 4. FC において式 (16) により ReLU に向かう勾配と，FC の行列 A とベクトル $\boldsymbol{b}$ の偏微分を計算する（関数 fc_bwd の引数 dEdy に，関数 relu_bwd で計算した dEdx を入れる）．

補題 12：ランダムシャッフル

確率的勾配降下法では，N 個の訓練データからランダムに n 個を選び出す操作を N/n 回行うことで，N 個全てのデータを 1 回ずつ使用することが想定されている．これをそのまま「ランダムに N 個の中から n 個を選ぶ」として実装して N/n 回行うと，1 回も選ばれないデータや，複数回選ばれるデータが発生することを避けるための処理が煩雑となる．そこで今回は「結果的に N 個全てのデータを使う」ことに着目して「並び替え」による実装を行う．

具体的には N 個の要素を持つ配列が与えられたとき，まず各要素をその添字番号によって初期化する．すなわち配列 x[] に対して x[i]=i とする．続いて

```
void relu(int n, const float * x, float * y) {
}
int main() {
  // 省略
  float * y = malloc(sizeof(float)*10);
  fc(10, 784, train_x, A_784x10, b_784x10, y);
  relu(10, y, y);
  print(1, 10, y);
  return 0;
}
```

出力

```
 1.6129  0.0000  2.0135  6.3752  0.0000  8.4595  0.0000  1.4323  1.2656  0.3464
```

図 5　relu()

```
void softmax(int n, const float * x, float * y) {
}
int main() {
  // 省略
  float * y = malloc(sizeof(float)*10);
  fc(10, 784, train_x, A_784x10, b_784x10, y);
  relu(10, y, y);
  softmax(10, y, y);
  print(1, 10, y);
  return 0;
}
```

出力

```
 0.0009  0.0002  0.0014  0.1101  0.0002  0.8853  0.0002  0.0008  0.0007  0.0003
```

図 6　softmax()

```
int inference3(const float * A, const float * b, const float * x) {
}
// テスト
int main() {
  // 省略
  float * y = malloc(sizeof(float)*10);
  int ans = inference3(A_784x10, b_784x10, train_x);
  printf("%d %d\n", ans, train_y[0]);
  return 0;
}
```

図 7　inference3()

```
#include "nn.h"
int main() {
  // 省略
  int i = 0;
  save_mnist_bmp(train_x + 784*i, "train_%05d.bmp", i);
  return 0;
}
```

図 8　save_mnist_bmp()

```
#include "nn.h"
int main() {
  // 省略
  int sum = 0;
  for(i=0 ; i<test_count ; i++) {
    if(inference3(A_784x10, b_784x10, test_x + i*width*height) == test_y[i]) {
      sum++;
    }
  }
  printf("%f%%\n", sum * 100.0 / test_count);
  return 0;
}
```

図 9　正解率

```
void softmaxwithloss_bwd(int n, const float * y, unsigned char t, float * dEdx);
```

図 10　softmaxwithloss_bwd()

```
void relu_bwd(int n, const float * x, const float * dEdy, float * dEdx);
```

図 11　relu_bwd()

```
void fc_bwd(int m, int n, const float * x, const float * dEdy, const float * A,
            float * dEdA, float * dEdb, float * dEdx);
```

図 12　fc_bwd()

```
 void backward3(const float * A, const float * b, const float * x, unsigned char t,
                float * y, float * dEdA, float * dEdb) {
}
int main() {
 // 省略
 float * y = malloc(sizeof(float)*10);
 float * dEdA = malloc(sizeof(float)*784*10);
 float * dEdb = malloc(sizeof(float)*10);
 backward3(A_784x10, b_784x10, train_x + 784*8, train_y[8], y, dEdA, dEdb);
 print(10, 784, dEdA);
 print(1, 10, dEdb);
 return 0;
}
```

出力

```
 (省略)
 0.0000 -0.0121  0.0000  0.0047  0.0000  0.0014  0.0000  0.0004  0.0032  0.0013
```

図 13　backward3()

```
void shuffle(int n, int * x) {
}
int main() {
 // 省略
 int * index = malloc(sizeof(int)*train_count);
 for(i=0 ; i<train_count ; i++) {
   index[i] = i;
 }
 shuffle(train_count, index);
 return 0;
}
```

図 14　shuffle()

各要素をランダムに選ばれた別の要素と入れ替える．つまり各 $i = 0, \ldots, N-1$ について，乱数によって $j \in [0 : N-1]$ を生成し，`x[i]` と `x[j]` を入れ替える．こうして得られた配列の先頭から順に n 個ずつ取り出すと，確率的勾配降下法が想定するように「N 個の訓練データからランダムに n 個を選び出す操作を N/n 回行うと N 個全てのデータを 1 回ずつ使用する」ことができる．

このような動作を行う関数として図 14 に示すものを作成せよ．

補題 13：損失関数

図 15 の仕様を満たすように式 (6) を実装せよ．ただし対数を計算する際に，引数が 0 となると出力が無限大となってしまい数値計算として破綻する．そこで式 (6) をそのまま実装するのではなく，$\boldsymbol{y}$ に微少量 $\epsilon = 1e-7$ を加えることでこの問題を回避すること．すなわち

$$E \approx -\sum_k t_k \log(y_k + \epsilon) \tag{17}$$

とせよ．

補題 14：補助関数

確率的勾配降下法を簡潔に実装するにあたり，図 16 のように配列の初期化，足し算，掛け算を行う補助関数を実装せよ．`add()` と `scale()` は $\frac{\partial E}{\partial A}$ および $\frac{\partial E}{\partial \boldsymbol{b}}$ の平均を計算し，さらに $A \leftarrow A - \eta\frac{\partial E}{\partial A}$ および $\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta\frac{\partial E}{\partial \boldsymbol{b}}$ を計算する際に使用できる．

補題 15：確率的勾配降下法による NN の学習

ここまでで実装した

- `fc()`
- `relu()`
- `softmax()`
- `inference3()`
- `softmaxwithloss_bwd()`
- `relu_bwd()`
- `fc_bwd()`
- `backward3()`
- `shuffle()`
- `cross_entropy_error()`
- `add()`
- `scale()`
- `init()`
- `rand_init()`

を組み合わせ，確率的勾配降下法による NN の学習を以下のように実装せよ．

1. エポック回数を決める．例として 10 回とする．
2. ミニバッチサイズ n を決める．例として $n = 100$ 個とする．同時に N/n によりバッチ学習回数が決まる．
3. 学習率 η を決める．例として $\eta = 0.1$ とする．
4. $A, \boldsymbol{b}$ を $[-1 : 1]$ の乱数で初期化する．
5. 以下をエポック回数だけ繰り返す．
   (a) $[0 : N-1]$ の値を持つ配列 `index` をランダムに並び替える．
   (b) 以下のミニバッチ学習を N/n 回繰り返す．
      i. 平均勾配 $\overline{\frac{\partial E}{\partial A}}, \overline{\frac{\partial E}{\partial \boldsymbol{b}}}$ を 0 で初期化する．
      ii. 配列 `index` から次の n 個を取り出す．
      iii. n 個の添字それぞれについて，対応する学習データと正解データを用いて $\frac{\partial E}{\partial A}, \frac{\partial E}{\partial \boldsymbol{b}}$ を計算し，$\overline{\frac{\partial E}{\partial A}}, \overline{\frac{\partial E}{\partial \boldsymbol{b}}}$ に加える．
      iv. $\overline{\frac{\partial E}{\partial A}}, \overline{\frac{\partial E}{\partial \boldsymbol{b}}}$ を n で割って平均勾配を得る．
      v. $A \leftarrow A - \eta\overline{\frac{\partial E}{\partial A}}$ および $\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta\overline{\frac{\partial E}{\partial \boldsymbol{b}}}$ により $A, \boldsymbol{b}$ を更新．
   (c) テストデータに対する損失関数および正解率を計算して表示．エポックごとに損失関数が低下し，正解率が向上していることを確認．

適当な初期化であっても，最初のエポックでテストデータに対する正解率は 35% 程度，10 回後には 80% 程度まで学習できるはずである．

補題 16：6 層 NN

ネットワークを 3 層から図 2 の 6 層に組み替えて `test_x`〜`test_x+(test_count-1)*784` を入力して推論を行い，正解 `test_y[0]`〜`test_y[test_count-1]` と比較することで正解率を計算せよ．正しく実装できている場合，正解率は 97.58% となるはずである．ただし係数は `nn.h` で定義された

- `A1_784_50_100_10` FC1 用の 50×784 行列
- `b1_784_50_100_10` FC1 用の 50 行ベクトル
- `A2_784_50_100_10` FC2 用の 100×50 行列
- `b2_784_50_100_10` FC2 用の 100 行ベクトル
- `A3_784_50_100_10` FC3 用の 10×100 行列
- `b3_784_50_100_10` FC3 用の 10 行ベクトル

を用いること．

補題 17：6 層 NN の学習

3 層の場合と同様に，6 層 NN 向けの誤差逆伝播と SGD を実装せよ．各 FC 層の係数 A_i, b_i を $[-1 : 1]$ の乱数によって初期化し，交差エントロピー

```
float cross_entropy_error(const float * y, int t) {
  return ... // 1行で書けるはず
}
```

図 15　cross_entropy_error()

```
void add(int n, const float * x, float * o) {
  // o[i] += x[i] を実行
}
void scale(int n, float x, float * o) {
  // o[i] *= x を実行
}
void init(int n, float x, float * o) {
  // o[i] = x を実行
}
void rand_init(int n, float * o) {
  // o[i] を [-1:1] の乱数で初期化
}
```

図 16　補助関数

誤差がエポックごとに低下することを確認すること．

補題 18：学習した係数のファイル保存

fread() および fwrite() を使用して，1 層分のパラメータを保存・読込する関数 save() および load() を作成せよ（図 17）．

補題 19：BMP 画像の読み込み

nn.h に BMP 画像の読込関数として load_mnist_bmp() が用意されている．図 18 に従い，実際に画像と自らが保存した学習パラメータをファイルから読み込んで，認識結果を表示せよ．

たとえば先の補題で行ったように，学習用の画像そのものを save_mnist_bmp() で保存し，それを認識させて正解ラベルと比較することができる．あるいは図 19 のように，任意のフォントで数字画像を作ることもできる．

また何も書かれていない画像，あえて数字以外を書いた画像などを入力したときの動作を調べ，どのような対処法が考えられるか検討してもよい．

その他のヒント

■実行すると nn.h のなかでエラーになる　プログラムを実行しているディレクトリ（カレントディレクトリ）に train-images-idx3-ubyte などのファイルが存在しないことが原因であることが多い．

■メモリエラー　メモリ関係のエラーが発生する，たとえば free() しようとするとエラーになる場合，「free() した直後にその変数に NULL を代入する」と，すでに free() した変数を使おうとしたときに直ぐにエラーになるので誤りに気が付きやすい．

たとえば free() を直接使うのではなく，図 20 のように，「free() すると NULL 代入を必ず行う」マクロを用意して，free() の代わりに必ずこちらを使うようにすると気が付きやすくなる．

ただしこの方法では「配列の範囲外を使ってしまっている」など他の原因には効果がない．より網羅的にメモリエラーを調べるには，valgrind を使うことができる（図 21）．ただし実行速度が極端に遅くなる．より効果的な使い方など，詳しくは valgrind のマニュアルを読むこと．

```
void save(const char * filename, int m, int n, const float * A, const float * b) {
}
void load(const char * filename, int m, int n, float * A, float * b) {
}
```

テスト（保存）

```
int main() {
  // 省略．配列 A1, b1, A2, b2, A3, b3 に学習できているとする．
  save("fc1.dat", 50, 784, A1, b1);
  save("fc2.dat", 100, 50, A2, b2);
  save("fc3.dat", 10, 100, A3, b3);
  return 0;
}
```

テスト（読込）

```
int main() {
  // 省略．配列 A1, b1, A2, b2, A3, b3 のメモリ領域が用意されているとする．
  load("fc1.dat", 50, 784, A1, b1);
  load("fc2.dat", 100, 50, A2, b2);
  load("fc3.dat", 10, 100, A3, b3);
  // 認識精度を計算
  return 0;
}
```

図 17　保存読込関数

```
#include "nn.h"
// 省略
int main(int argc, char * argv[]) {
  float * A1 = malloc(sizeof(float)*784*50);
  float * b1 = malloc(sizeof(float)*50);
  float * A2 = malloc(sizeof(float)*50*100);
  float * b2 = malloc(sizeof(float)*100);
  float * A3 = malloc(sizeof(float)*100*10);
  float * b3 = malloc(sizeof(float)*10);
  float * x = load_mnist_bmp(argv[4]);
  load(argv[1], 50, 784, A1, b1);
  load(argv[2], 100, 50, A2, b2);
  load(argv[3], 10, 100, A3, b3);
  return 0;
}
```

図 18　load_mnist_bmp()

convert コマンド（Imagemagick パッケージ）をインストール

```
$ sudo apt update && sudo apt install imagemagick
```

デフォルトのフォントで数字 8 を描画し，a.bmp に保存

```
$ convert -size 28x28 xc:black -pointsize 24 -fill white -draw "text 7,26 '8'" a.bmp
```

指定したフォントで数字 6 を描画し，b.bmp に保存

```
$ convert -size 28x28 xc:black -pointsize 24 -fill white -font /c/Windows/Fonts/impact.ttf
-draw "text 7,26 '6'" b.bmp
```

図 19　convert による数字画像の生成

```
#include <signal.h>
#include "nn.h"
#define MY_FREE(p)    \
    {                 \
        if (p)        \
        {             \
            free(p);  \
            p = NULL; \
        }             \
        else          \
        {             \
            fprintf(stderr, "double free of ‘‘" #p "’’ at line %d\n", __LINE__); \
            raise(SIGTRAP); \
        }             \
    }
void backward6((省略)) {
    float * x = (float *)malloc(10 * sizeof(float));
    MY_FREE(x);  // x に NULL が自動的に代入される
    x[0] = 10;   // すでに x が NULL なのでエラーになる（デバッガが一時停止する）
    MY_FREE(x);  // もう一度 free() しようとするとエラーになる（デバッガが一時停止する）
}
```

図 20　free() と NULL 代入を同時に行うためのマクロ

valgrind コマンドをインストール

```
$ sudo apt update && sudo apt install valgrind
```

valgrind コマンドを使ってエラーをチェックするには，通常の実行コマンドの前に valgrind をつけるだけ．デバッグビルドしておく必要がある．最初のエラーが表示されたら即座に Ctrl-C で止めて，その箇所を直すのが確実.

```
$ valgrind ./a.out
```

図 21　valgrind のインストールと実行