メディア情報処理 (パターン認識) レポート

田中智也 (1615092T)

提出日 2018 年 12 月 26 日

1　理論とプログラム仕様

以下に, プログラムの処理の流れを記述する (大括弧内は当該処理に対応するソースコードのファイル名).

1. 重みの初期値を設定する.[initialize_weight.c]
2. class1.dat, class2.dat, test.dat からデータを読み込む.[read_data.c]
3. 訓練データを用いて, ニューラルネットワークにおける最適な重み係数を求める.[neural_net.c]
4. テストデータを用いて, モデルの精度を検証する [neural_net.c].

次に, ニューラルネットワークを用いた分類アルゴリズムの流れを記述する.

変数の定義

着目する階層のユニットに対して, $j = 1, 2, \cdots, n$ と番号をつける. また, 1 階層前のユニットに対しては, $i = 1, 2, \cdots, n$, 1 階層後のユニットに対しては, $k = 1, 2, \cdots, K$ と番号をつける. その上で,

$f :=$ シグモイド関数
$w_{ij} :=$ ユニット i からユニット j へのエッジに対応する重み
$u_{pj} :=$ 入力信号 p に対するユニット j の総入力
$s_{pj} :=$ 出力層における, 入力信号 p に対する教師信号
$v_{pj} := f(u_{pj})$
$J :=$ 誤差関数
$\rho :=$ 学習率

とする. ただし,

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

$$J(v, s) = \frac{1}{2} \sum_{p} \sum_{j} (v_{pj} - s_{pj})^2 \tag{2}$$

である. また, 入力信号 p に対する誤差関数 $J(v, s) \frac{1}{2} \sum_j (v_{pj} - s_{pj})^2$ を J_p と表記する.

理論と関数実装

定義した変数を用いて, ユニット i からユニット j への信号の伝播は,

$$u_{jp} = \sum_i w_{ij} v_{pi} \tag{3}$$

$$v_{jp} = f(u_{pj}) \tag{4}$$

とかける. 入力信号 p が出力層まで伝播したとき, 誤差を J_p を用いて評価することができる. 以上の処理を, 関数 *forward_propagation* にまとめた. 次に, 誤差を入力層までフィードバックし, 重みを修正する. 各入力信号 p に対する重みの修正式は, 式 (3) の w_{ij} に関する微分が, v_{pi} であることを用いて,

$$\begin{aligned}
\hat{w}_{ij} &= w_{ij} - \rho \frac{\partial J_p}{\partial w_{ij}} \\
&= w_{ij} - \frac{\partial J_p}{\partial u_{pj}} \frac{\partial u_{pj}}{\partial w_{ij}} \\
&= w_{ij} - \frac{\partial J_p}{\partial u_{pj}} v_{pi}
\end{aligned} \tag{5}$$

ここで新たに右辺第二項の $\frac{\partial J_p}{\partial u_{pj}}$ を e_{pj} とすると,

$$\begin{aligned}
\frac{\partial v_{pj}}{\partial u_{pj}} &= \frac{\partial f(u_{pj})}{\partial u_{pj}} \\
&= (1 - f(u_{pj})) f(u_{pj}) \\
&= (1 - v_{pj}) v_{pj}
\end{aligned} \tag{6}$$

となることを用いて,

$$\begin{aligned}
e_{pj} &= \frac{\partial J_p}{\partial u_{pj}} \\
&= \frac{\partial J_p}{\partial v_{pj}} \frac{\partial v_{pj}}{\partial u_{pj}} \\
&= \frac{\partial J_p}{\partial v_{pj}} (1 - v_{pj}) v_{pj}
\end{aligned} \tag{7}$$

最右辺の $\frac{\partial J_p}{\partial v_{pj}}$ については, 出力層では式 (8) のように表される.

$$\frac{\partial J_p}{\partial v_{pj}} = v_{pj} - s_{pj} \tag{8}$$

中間層では, 式 (3) の v_{ip} に関する微分が, w_{ij} であることを用いて,

$$\begin{aligned}
\frac{\partial J_p}{\partial v_{pj}} &= \sum_k \frac{\partial J_p}{\partial u_{pk}} \frac{\partial u_{pk}}{\partial v_{pj}} \\
&= \sum_k e_{pk} w_{jk}
\end{aligned} \tag{9}$$

となる. ここで, 右辺が $\frac{\partial J_p}{\partial u_{pk}} \frac{\partial u_{pk}}{\partial v_{pj}}$ の総和となっているのは, v_{pj} が次の層の全ユニットに作用するからである. 式 (7) 〜 (9) をまとめると,

$$e_{pj} = \begin{cases} (v_{pj} - s_{pj})(1 - v_{pj}) v_{pj} & \text{(出力層)} \\ (\sum_k e_{pk} w_{jk})(1 - v_{pj}) v_{pj} & \text{(中間層)} \end{cases} \tag{10}$$

式 (10) の処理を, 関数 *back_propagation* にまとめた. 最後に重みを更新することになるが, 更新式は式 (5), (10) を用いて,

$$\hat{w}_{ij} = w_{ij} - e_{pj}v_{pi} \tag{11}$$

となる. 式 (11) は関数 *weight_update* にまとめた.
プログラム中では, 定義した各変数は,

$w_{ij} \rightarrow W1[i][j]$(入力層と隠れ層の間の重み), $W2[i][j]$(隠れ層と出力層の間の重み)
$s_{pj} \rightarrow teacher[p][j]$
$v_{pj} \rightarrow input[p][j], hidden[j], output[p][j]$
$e_{pj} \rightarrow delta\_hidden[j], delta\_output[j]$
$J \rightarrow total\_error$
$\rho \rightarrow rho$

と対応づけている. また, 評価関数値の値の変化量が微小な値 ϵ より小さくなることを終了条件とし, 各層にはバイアス (ユニットの値が常に 1) を導入している.

2 実行方法

基本の流れ

```
$make mkdata
$make
$make view
$make clean
```

各種機能

データの用意
$make load : 図 1 に示すデータとテストデータの生成
$make mkdata : 図 2 に示すデータとテストデータの生成
$make mkdata_nonlinear : 図 3 に示すデータとテストデータの生成
$make mkdata_circle : 図 4 に示すデータとテストデータの生成

コンパイルと実行
$make : プログラムのコンパイルと実行.
$make view : 読み込んだデータと結果をグラフとして出力

削除
$make clean : オブジェクトファイルと出力データを削除
$make rmdata : 作成したデータを削除

※隠れ層の数や出力層の数などの指定方法は作成していただいたサンプルプログラムに従っています.
※テストデータの形式は 1〜n-1 列目: データ, クラス: n+1 列目 (課題テストデータに対しては正解率が正

しく表示されないが, モデル構築は可能)

3　結果

ニューラルネットワークを用いた分類器には, 図 1〜4 に示す 4 つのデータを訓練データとして与えた. 図 1,2 の示すデータは線形分離可能であり, 図 3,4 の示すデータは線形分離不可能である. 図 1 のデータは課題として与えられたデータであり, 図 2〜4 については独自に作成したものである. 図 2〜4 が示すデータの性質について簡単に表 1 にまとめておく.

表 1: 各データ特性

データ 2	課題データと各クラスの中心点はほぼ同じであるが, 分散が課題データより大きい.
データ 3	課題データより分散が大きく, かつ, 1 クラスあたり 2 つのクラスタを有している.
データ 4	クラス 1 は半径 2.5 の円内に, クラス 2 は半径 5 の円の中に入っていて, かつ, 半径 3 の円の中に属していない.

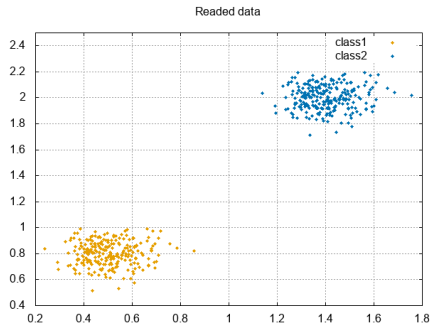


図 1: データ 1

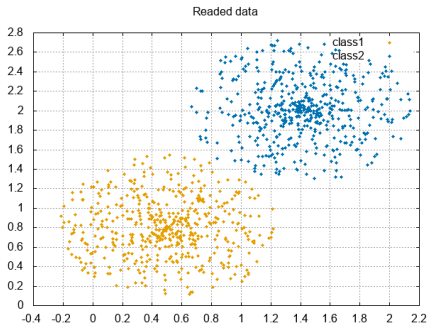


図 2: データ 1

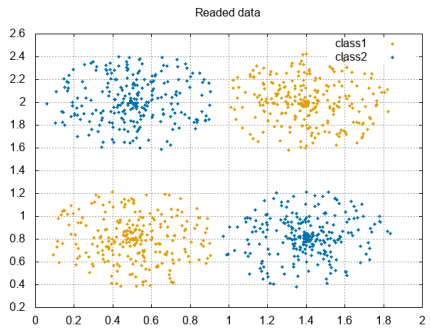


図 3: データ 3

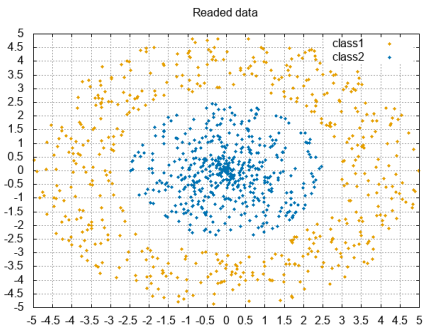


図 4: データ 4

以上のデータを, 図 1 が示すデータについては訓練データ 250 個, テストデータ 50 個で, 図 2〜4 が示すデータについては訓練データ 500 個, テストデータ 400 個でモデル構築とモデル検証を行った. 結果を図 5 〜 図 8 に示す. 各結果の左の図は各訓練データに対する出力値であり, 右の図は誤差関数値の推移を表している. ただし, 学習率 ρ は 0.1, 終了条件 ϵ は 10^{-8}, 隠れ層のユニット数は 4 とした.

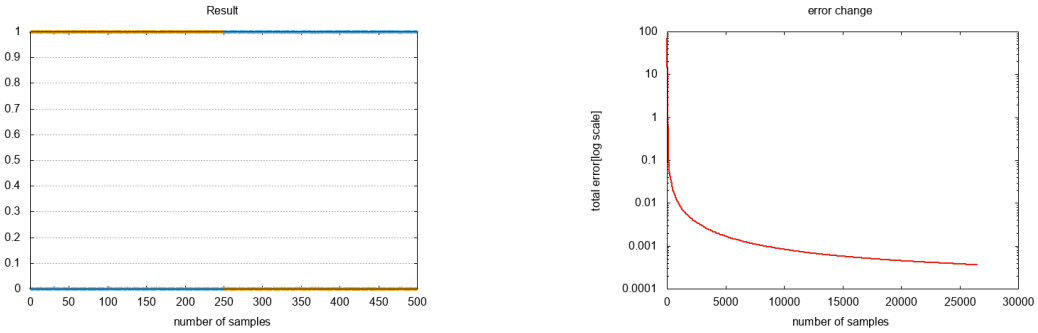


図 5: データ 1 を用いたモデル構築

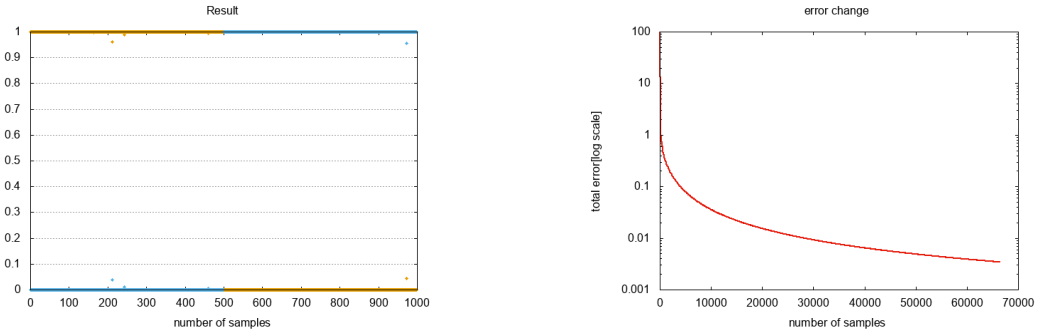


図 6: データ 2 を用いたモデル構築

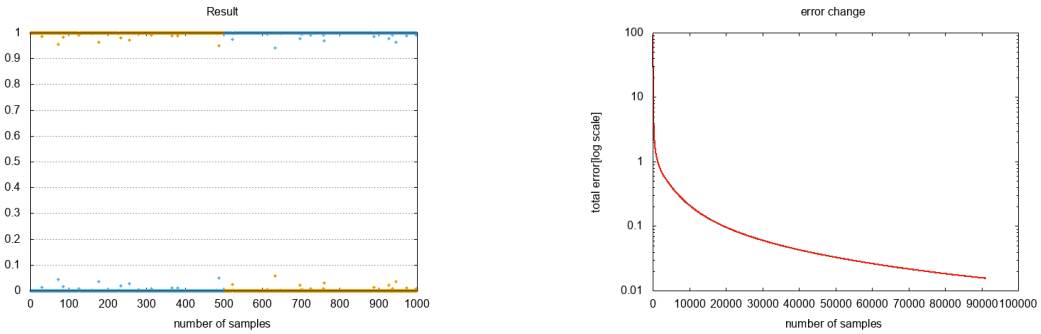
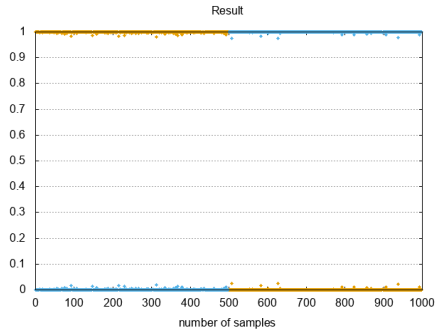


図 7: データ 3 を用いたモデル構築



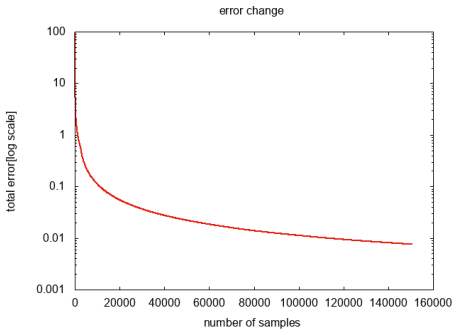


図 8: データ 4 を用いたモデル構築

各サンプルに対する出力値を見ると, モデルは学習データに十分適合していると捉えられる. 各モデルに対して, 訓練データと同様の方法で作成したテストデータを与えると, 正解率 (正しく出力されたデータ数/全データ数) は, すべて 1.0 となった. 訓練データを 250 とすると, 正解率が少し小さくなったが, 0.97 よりは大きくなった. よって, 過学習も起きてなさそうだ. 以上の結果より, 線形分離可能なデータはもちろん, 線形分離不可能なデータに対しても正確に分類することができているといえる.
ただ, 図 5, 図 8 の右図より, データ 4 では, 収束するのにデータ 1 の 5 倍近くのイテレーション数が必要となっている. そこで, 次の式のように学習率を徐々に下げていくと実行時間は $51s \rightarrow 10s$ に減少し, イテレーション数も $166904 \rightarrow 33975$ に減少した. このときも正解率は 1.0 であった.

$$\rho = \rho_0 * 0.99^{[\frac{1+t}{k}]} (k : \text{学習率を下げるステップ間隔}, t : \text{ステップ数}, \rho_0 : \text{初期値})$$

よって, 学習率を徐々にさげていくことにより, 収束性が向上することがわかる.

4 補足:隠れ層の機能についての考察

データ 3 とデータ 4 は線形分離不可能と書いたが, 線形分離不可能なのは 2 次元の世界での話で, 高次元空間に射影すれば線形分離可能である. データ 3 については, 左上, 右上, 左下, 右下の各クラスタのデータをそれぞれ, (0,1), (1,1), (1,0), (0,0) に射影し, クラス 1 が 0 をクラス 2 が 1 を表していると考えれば, XOR 問題に帰着される. XOR 問題は例えば

$$\phi(x_1, x_2) = (x_1, x_2, x_1 \wedge x_2) \tag{12}$$

という射影を用いれば, 3 次元空間で線形に分離できる. データ 4 については,

$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2) \tag{13}$$

という射影を用いれば同様に線形分離可能となる. データ 4 について詳しく見てみよう. データ 4 の射影関数は, 式 (14) のような線形変換を行い

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 + x_2 \end{bmatrix} \tag{14}$$

それぞれの出力に対して, 活性化関数 $f(x) = x, f(x) = x$, ある活性化関数 (思いつかない... 射影関数と線形変換を工夫すれば見つかるはず) を施すことで表現することができる. 線形分離可能なデータに射影してしまえば, 単純パーセプトロンと同じ原理で, それぞれの出力に適当な重み付けをおこなうことで, 分類を行うことができる.

実験では, 3 層のニューラルネットを用いたが,

1. 単純パーセプトロンが線形分離可能なデータしか分類できないこと
2. 実験で, 線形分離不可能なデータでも完全に分類できたこと

を考えれば, 以上で議論したように, 入力層から隠れ層 (出力層の手前の層) の段階で, 線形分離不可能なデータを線形分離可能なデータに射影することができていると考えられる. つまり, 隠れ層は線形分離不可能なデータを, (正解率が 1 でない場合はある程度までだが) 線形分離可能なデータに変換する機能をもつと捉えることができる.

5　課題のテストデータのクラス

最後に, 課題のテストデータに対する出力値を図 9 に示す. 結果より, テストデータはすべてクラス 1 に属することがわかる.

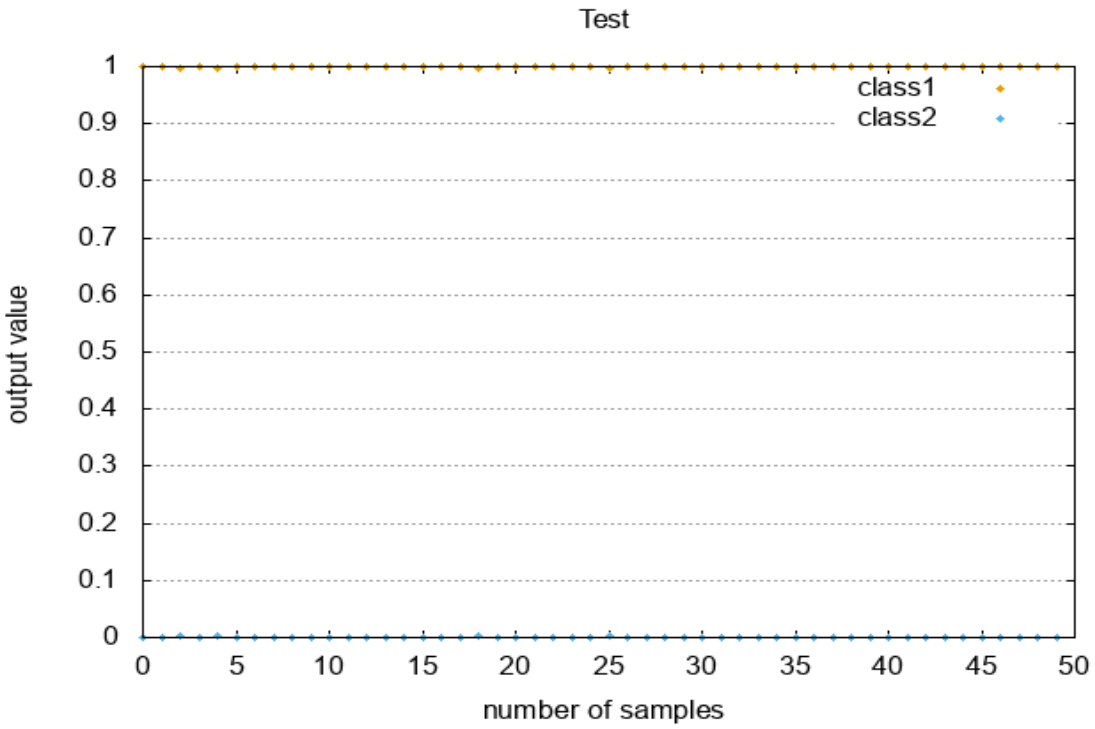


図 9: 課題のテストデータに対する出力値

参考文献

[1] Sebastian Raschka『Python 機械学習プログラミング』(2016) インプレス

[2] 斎藤康毅『ゼロから作る Deep Learning』(2016) オライリー・ジャパン

[3] メディア情報処理講義資料 (滝口先生作成)(http://www.me.cs.scitec.kobe-u.ac.jp/~takigu/class/media/work/nn.pdf)

[4] 総合演習 1 講義資料 (江口先生作成)(http://www.prmir.scitec.kobe-u.ac.jp/local/enshuA2018/enshuA1-2-2018.html#7th)