知能システム前期末レポート

15422 篠田拓樹

2019 年 8 月 8 日

1　XOR を学習するニューラルネットワーク

XOR を学習するニューラルネットワークを作成した．リスト 1 に示すプログラムを用いて実行を行った．

XOR の各入力に対して学習がうまくいっていることを確認した．

また，拡張として行列演算によって前向き，後ろ向きの計算を行うリスト 2 のプログラムも作成した．なお，リスト 1 と同じ重みを用いて学習を行い，結果が同じになることを確認している．同様に学習がうまくいっていることを確認した．

学習過程の誤差の変化を図 1 に示す．

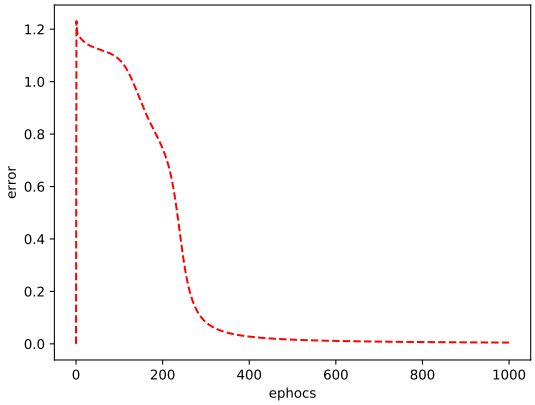

図 1　XOR 学習時の誤差変化

2　魚を識別するニューラルネットワーク

作成したプログラムをリスト 3 に示す．今回は，トレーニングデータ A と B を読み込んでラベル付けを行い，シャフルしトレーニングデータとテストデータに分けた．ネットワークは，XOR と同様のものを使い，1，0 での分類を行った．

fish の学習がうまく行っていることをテストデータで確認した．テストデータでは，$\frac{9}{10}(90\ \%)$ の精度を記録した．

学習過程の誤差の変化を図 2 に示す．

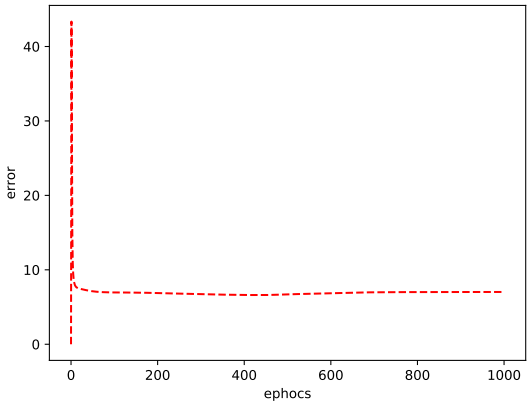

図 2　fishA,B 学習時の誤差変化

3　プログラムリスト

リスト 1　sampleBP.c

```
/*
 * NeuralNetwork For XOR
 *
 * Input layer:  2
 * Hidden layer: 2
 * Output layer: 1
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define EPSILON 4.0
#define ETA 0.1
#define TIMES 1000
#define INIT_WEIGHT 0.3

double randNum(void)
{
  return ((double)rand()/RAND_MAX-0.5)*2.0*
   INIT_WEIGHT;
}

double sigmoid(double x)
{
  return 1/(1+exp(-1*EPSILON*x));
}

int main(void)
{
  double data[4][3] = {
    {0.0, 0.0, 0.0},
    {0.0, 1.0, 1.0},
    {1.0, 0.0, 1.0},
    {1.0, 1.0, 0.0}
  };
  double wbd, wbe, wcd, wce, wab, wac;
  double offb, offc, offa;
  double outd, oute, outb, outc, outa;
  double xb, xc, xa;
  double deltab, deltac, deltaa;
  int r;
  double error;
  double errorSum;
  int times;
  int seed;
  FILE *fp;

  fp = fopen("error.dat", "w");
  if (fp==NULL) {
    printf("can't open file.\n");
    exit(1);
  }

  //seed = (unsigned int)time(NULL);
  //printf("seed = %d\n", seed);
  seed = 0;
  srand(seed);

  wbd = randNum();
  wbe =  randNum();

  wcd =  randNum();
  wce =  randNum();

  wab =  randNum();
  wac =  randNum();

  offb = randNum();
  offc =  randNum();
  offa =  randNum();

  for(times=0;times<TIMES; times++) {

    errorSum = 0.0;

    for(r=0; r<4; r++) {

      /* ------------ */
      /* Feedforward */
      /* ------------ */

      /* Input layer output */
      outd = data[r][0];
      oute = data[r][1];

      /* Hidden layer output */
      xb = wbd*outd + wbe*oute + offb;
```

```
        outb = sigmoid(xb);

        xc = wcd*outd + wce*oute + offc;
        outc = sigmoid(xc);

        /* Output layer output */
        xa = wab*outb + wac*outc + offa;
        outa = sigmoid(xa);

        if(times==TIMES-1) {
          printf("[%d]=%.10f, (%f)\n", r, outa,
            data[r][2]);
        }

        /* ---------------- */
        /* Back Propagation */
        /* ---------------- */
        error = ((outa-data[r][2])*(outa-data[r
          ][2]));
        errorSum += error;

        /*
         *
         * ここに更新式を書く
         *
         * deltaa = ...
         * wab = wab + ...
         *
         */
        deltaa = (outa - data[r][2])*EPSILON
          *(1-outa)*outa;
        deltab = (deltaa * wab)*EPSILON*(1-outb
          )*outb;
        deltac = (deltaa * wac)*EPSILON*(1-outc
          )*outc;

        wab = wab - ETA * deltaa * outb;
        wac = wac - ETA * deltaa * outc;
        offa = offa - ETA * deltaa;

        wbd = wbd - ETA * deltab * outd;
        wbe = wbe - ETA * deltab * oute;
        offb = offb - ETA * deltab;

        wcd = wcd - ETA * deltac * outd;
        wce = wce - ETA * deltac * oute;
        offc = offc - ETA * deltac;
      }
      printf("errorSum = %f\n", errorSum/4.0);
      fprintf(fp, "%f\n", errorSum/4.0);
      //printf(" wab = %f\n wac = %f\n offa=%f\
        n",wab,wac,offa);
      //printf(" wbd = %f\n wbe = %f\n offb=%f\
        n",wbd,wbe,offb);
      //printf(" wcd = %f\n wce = %f\n offc=%f\
        n",wcd,wce,offc);
    }
    printf(" wab = %f\n wac = %f\n offa=%f\n",
      wab,wac,offa);
    printf(" wbd = %f\n wbe = %f\n offb=%f\n",
      wbd,wbe,offb);
    printf(" wcd = %f\n wce = %f\n offc=%f\n",
      wcd,wce,offc);


    fclose(fp);

    return 0;
}
```

リスト 2　xor_newral.py

```
import numpy as np
import matplotlib.pyplot as plt

class Newral:
    def __init__(self, in_newrons,
      mid_newrons, out_newrons):
        self.eps = 4
        self.errors = np.zeros((1,out_newrons
          ))
        self.mid_weight = np.array
          ([[-0.133335,-0.098866],
                                          [
                                           0.204113,
                                           0.169860],
                                          [-0.063370,
                                           0.179064]])
        #np.random.random_sample((in_newrons
          + 1, mid_newrons))
        self.out_weight = np.array
          ([[0.160938],
                                          [0.246988],
                                          [-0.181469]])
        #np.random.random_sample((mid_newrons
          + 1, out_newrons))

    """
    シグモイド関数
    ベクトルを計算するために関数を
      vectorizeで宣言
    """
    def sigmoid(self, vec):
        return np.vectorize(lambda x : 1.0 /
          (1.0 + np.exp(-1 * self.eps * x)))(
          vec)

    """
    シグモイド関数の微分関数
    ベクトルを計算するために関数を
      vectorizeで宣言
    """
    def grad_sigmoid(self, out_vec):
        return np.vectorize(lambda x : self.
          eps * (1 - x) * x)(out_vec)

    def mean_sqard_error(self, out_vec, t_vec
      ):
        return np.vectorize(lambda x,y : (x -
          y)**2)(out_vec, t_vec)

    """
    前向きの計算

    行列で書き下すと分かるはず…
    仮想ニューロンを導入しているので(1,x_1,
      x_2...)^Tと
    の掛け算になる.
    """
    def forward(self, x):
        out_mid = self.sigmoid(np.r_[np.array
          ([1]), x].dot(self.mid_weight))
        out_out = self.sigmoid(np.r_[np.array
          ([1]), out_mid].dot(self.out_weight
          ))

        return (out_mid, out_out)

    """
    学習を行う
    """
    def train(self, x, t, eta, times):
        for i in range(times):
            total_error = 0
            for j, k in zip(x, t):
                total_error += N.BP(j, k, eta
                  )
            self.errors = np.append(self.
              errors, total_error.reshape
              (1,-1), axis=0)

```

```
    # 1パターン分の学習を行う
    def BP(self, x, t, eta):
        # 前向きの計算
        out_mid, out_out = self.forward(x)
        # 誤差の計算
        error = self.mean_sqard_error(out_out, t)
        # デルタを先に計算する
        out_delta = (out_out - t) * self.grad_sigmoid(out_out)
        mid_delta = self.grad_sigmoid(out_mid) * (self.out_weight[1:,:].dot(out_delta))
        # 更新量を計算
        update_out = np.r_[np.array([1]), out_mid].reshape(-1,1).dot(out_delta.reshape(1,-1))
        update_mid = np.r_[np.array([1]), x].reshape(-1,1).dot(mid_delta.reshape(1,-1))
        # 更新
        self.out_weight -= eta * update_out
        self.mid_weight -= eta * update_mid

        return error

    """
    誤差の変化を確認するグラフ
    """
    def error_graph(self):
        #plt.figure(figsize=(10,20))
        plt.xlabel("ephocs")
        plt.ylabel("error")
        plt.plot(self.errors[:,0],"r-")
        #plt.show()
        plt.savefig("../report/img/error1.pdf")

if __name__ == "__main__":
    # 各層のニューロン数
    in_newrons, mid_newrons, out_newrons = (2, 2, 1)
    # 入力
    x = np.array([[0,0],[0,1],[1,0],[1,1]])
    # 出力
    t = np.array([[0],[1],[1],[0]])

    # ニューラルネット
    N = Newral(in_newrons, mid_newrons, out_newrons)
    # 訓練
    N.train(x, t, eta = 0.1, times = 1000)
    # 誤差グラフ
    N.error_graph()

    # 前向き計算で確認
    for i,j in zip(x,t):
        o1,o2 = N.forward(i)
        print(i,o2,j)
```

リスト 3　fish_newral.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random

"""
3層全結合型ニューラルネットワーク
を作成するクラス
"""
class Newral:
    """
    コンストラクタ
    """
    def __init__(self, in_newrons, mid_newrons, out_newrons):
        # シグモイド関数のepsilon
        self.eps = 4
        # 誤差記録用
        self.errors = np.zeros((1,out_newrons))
        # 中間層用重み
        #self.mid_weight = np.random.random_sample((in_newrons + 1, mid_newrons))
        self.mid_weight = np.array([[-0.133335,-0.098866],
                                    [ 0.204113, 0.169860],
                                    [-0.063370, 0.179064]])

        # 出力層用重み
        #self.out_weight = np.random.random_sample((mid_newrons + 1, out_newrons))
        self.out_weight = np.array([[0.160938],
                                    [0.246988],
                                    [-0.181469]])

    """
    シグモイド関数
    ベクトルを計算するために関数を
    vectorizeで宣言
    """
    def sigmoid(self, vec):
        return np.vectorize(lambda x : 1.0 / (1.0 + np.exp(-1 * self.eps * x)))(vec)

    """
    シグモイド関数の微分関数
    ベクトルを計算するために関数を
    vectorizeで宣言
    """
    def grad_sigmoid(self, out_vec):
        return np.vectorize(lambda x : self.eps * (1 - x) * x)(out_vec)

    """
    2乗和誤差
    """
    def mean_sqard_error(self, out_vec, t_vec):
        return np.vectorize(lambda x,y : (x - y)**2)(out_vec, t_vec)

    """
    前向きの計算

    行列で書き下すと分かるはず…
    仮想ニューロンを導入しているので(1,x_1,x_2...)^Tと
    の掛け算になる.
    """
    def forward(self, x):
        # 中間層の出力
        out_mid = self.sigmoid(np.r_[np.array([1]), x].dot(self.mid_weight))
        # 出力層の出力
        out_out = self.sigmoid(np.r_[np.array([1]), out_mid].dot(self.out_weight))

        return (out_mid, out_out)

    def predict(self, x, t):
        right = 0

        for i,j in zip(x,t):
            mid,out = self.forward(i)
            print(i,j)
            if out >= 0.5 : ans = 1
            else : ans = 0
            print(out,ans)

            if ans == j : right += 1
```

```
77
78              return right / len(t)
79          """
80          学習を行う
81          """
82          def train(self, x, t, eta, times):
83              for i in range(times):
84                  # 誤差の合計値
85                  total_error = 0
86                  # 全パターン学習
87                  for j, k in zip(x, t):
88                      total_error += N.BP(j, k, eta
                            )
89                  # 誤差を記録
90                  print("error : ",total_error)
91                  self.errors = np.append(self.
                        errors, total_error.reshape
                        (1,-1), axis=0)
92
93          """ 1パターン分の学習を行う """
94          def BP(self, x, t, eta):
95              # 前向きの計算
96              out_mid, out_out = self.forward(x)
97              # 誤差の計算
98              error = self.mean_sqard_error(out_out
                    , t)
99              # デルタを先に計算する
100             out_delta = (out_out - t) * self.
                    grad_sigmoid(out_out)
101             mid_delta = self.grad_sigmoid(out_mid
                    ) * (self.out_weight[1:,:].dot(
                    out_delta))
102             # 更新量を計算
103             update_out = np.r_[np.array([1]),
                    out_mid].reshape(-1,1).dot(out_delta
                    .reshape(1,-1))
104             update_mid = np.r_[np.array([1]), x].
                    reshape(-1,1).dot(mid_delta.reshape
                    (1,-1))
105             # 更新
106             self.out_weight -= eta * update_out
107             self.mid_weight -= eta * update_mid
108
109             return error
110
111         """
112         誤差の変化を確認するグラフ
113         """
114         def error_graph(self):
115             #plt.figure(figsize=(10,20))
116             plt.xlabel("ephocs")
117             plt.ylabel("error")
118             plt.plot(self.errors[:,0],"r-")
119             #plt.show()
120             plt.savefig("../report/img/error2.pdf
                    ")
121
122 def make_dataset():
123     # ファイル読込
124     x_A = pd.read_csv("../data/fishA.train",
            sep='\s+', header=None).values
125     x_B = pd.read_csv("../data/fishB.train",
            sep='\s+', header=None).values
126     # 入力と出力を結合
127     xy_A = np.insert(x_A, 2, 1,axis=1) #
            Aは教師1
128     xy_B = np.insert(x_B, 2, 0,axis=1) #
            Bは教師0
129     xy = np.vstack((xy_A,xy_B)) # 結合
130     # シャッフル
131     index = list(range(xy.shape[0]))
132     random.shuffle(index)
133     dataset = xy[index]
134     print("data set\n",dataset[:10])
135     # テストとトレインに分ける
136     train, test = np.split(dataset, [int(
            dataset.shape[0] * 0.9)])
137
138     return (train, test)
139
140 if __name__ == "__main__":
141     # 各層のニューロン数
142     in_newrons, mid_newrons, out_newrons =
            (2, 2, 1)
143     # データセット作成
144     train,test = make_dataset()
145     # 入力
146     x = train[:,:-1]
147     # 出力
148     t = train[:,-1]
149
150     # ニューラルネット
151     N = Newral(in_newrons, mid_newrons,
            out_newrons)
152     # 訓練
153     N.train(x, t, eta = 0.1, times = 1000)
154     # 誤差グラフ
155     N.error_graph()
156
157     # 前向き計算で確認
158     rate = N.predict(test[:,:-1],test[:,-1])
159     print(rate)
```