神经网络

贾成君 自46 2014011552

目录

[神经网络 1](#_Toc478925012)

[数据内容的显示 1](#_Toc478925013)

[样本训练神经网络 1](#_Toc478925014)

[数据分类 2](#_Toc478925015)

[附录1：神经网络训练结果 3](#_Toc478925016)

[附录2：源代码 8](#_Toc478925017)

# 数据内容的显示

抽取0-9的图片各一张，显示如下：



# 样本训练神经网络

把数据进行归一化处理并使用不同的隐藏节点进行训练，然后进行测试；得到的结果整理如下（更加详细的信息见后文附录）：

|  |  |  |  |
| --- | --- | --- | --- |
| 隐藏节点 | 训练时间 | 迭代次数 | 样本测试错误率 |
| 5 | 2min | 386 | 11.7% |
| 10 | 1min27s | 213 | 8.6% |
| 20 | 58s | 119 | 6.6% |
| 40 | 1min12s | 93 | 5.5% |
| 100 | 2min45s | 101 | 4.1% |

整理如下图：



从图形，可以看出：随着隐藏节点的增加，错误率下降，迭代次数整体呈现下降，训练用时先减小后增大。当隐藏节点增加时，模型的可变参数数目增多，自由度增大，对于数据的拟合估计的更好，能够使得错误率减小。而每次训练带来的更新更多（参数更多），收敛的更快，训练模型能够经过较小的次数而收敛，但是隐藏节点的增多使得每次迭代更新的工作量大大增加，每次更新的速度下降，整体使得收敛用时先减后增。

# 数据分类

针对隐藏结点为100的训练结果，可以看出，混淆最严重的应当是数字9和数字4（第5列和第10列）。总共的训练样本为：‘4’的训练数目：5842，‘9’的训练数目5945；测试样本里：‘4’的测试数目982，‘9’的测试数目1009；对于这两类进行分类（Logistics[[1]](#footnote-1)和Fisher分类）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ‘4’误判‘9’ | ‘9’误判‘4’ | 总错误率 | 用时 |
| 神经网络 | 25，2.55%[[2]](#footnote-2) | 14，1.39% | 39，1.96% | 175s |
| Logistics迭代上限100次 | 35，3.56% | 35，3.47% | 70，3.52% | 17s |
| Logistics迭代上限500次 | 23，2.34% | 39，3.87% | 62，3.11% | 94s |
| Logistics迭代上限1000次 | 33，3.36% | 29，2.87% | 62，3.11% | 214s |
| Fisher判据[[3]](#footnote-3) | 42，4.28% | 31，3.07% | 73，3.67% | 61s |

根据结果来看，三种训练的分类效果都还不错，但是神经网络的结果明显更优（错误率更小一些），而且从运行的效率来看，神经网络的效率也比较高[[4]](#footnote-4)；这说明了神经网络对于非线性函数的拟合效果比较理想而且能够达到较好的预测效果，而且运算量相比其他算法并没有特别大。

# 附录1：神经网络训练结果

|  |  |
| --- | --- |
| 隐藏节点 | 5，训练时间2min左右，迭代次数：386 |
| 测试样本混淆矩阵 |  |
| 训练误差曲线 |  |

|  |  |
| --- | --- |
| 隐藏节点 | 10，训练时间1min27s左右，迭代次数：213 |
| 测试样本混淆矩阵 |  |
| 训练误差曲线 |  |

|  |  |
| --- | --- |
| 隐藏节点 | 20，训练时间58s左右，迭代次数：119 |
| 测试样本混淆矩阵 |  |
| 训练误差曲线 |  |

|  |  |
| --- | --- |
| 隐藏节点 | 40，训练时间1min12s左右，迭代次数：93 |
| 测试样本混淆矩阵 |  |
| 训练误差曲线 |  |

|  |  |
| --- | --- |
| 隐藏节点 | 100，训练时间2min45s左右，迭代次数：101 |
| 测试样本混淆矩阵 |  |
| 训练误差曲线 |  |

# 附录2：源代码

源码可以从<https://github.com/ChengjunJia/PatternRecognition> 下载（对于logistic regression进行了一些算法优化[使用向量计算方式]）

|  |
| --- |
| %% Pattern Recognize ---- Neural Pattern Recognize  load**(**'MNIST.mat'**);**  %Data Normalize  testX1 **=** reshape**(**testX**,**size**(**testX**,**1**),[])** **./**255**;**  trainX1 **=** reshape**(**trainX**,** size**(**trainX**,**1**),[])** **./**255**;**  im\_side **=** 28**;**  %% Show the image  % figure,  % for m = 1:10  % loc = find(trainY(:,m));  % aimX = trainX(loc,:,:);  % loc2 = find( max(reshape(aimX,[],im\_side\*im\_side),[],2) >=150,1);  % subplot(2,5,m),  % image = reshape( aimX(loc2,:,:),im\_side,im\_side );  % imshow(image);  % end  %% Choose the '4' and '9', do the classify, '4'->0, '9'->1  testX4 **=** testX1**(** testY**(:,**5**)==**1**,:** **);**  testX9 **=** testX1**(** testY**(:,**10**)==**1**,:** **);**  trainX4 **=** trainX1**(** trainY**(:,**5**)==**1**,:** **);**  trainX9 **=** trainX1**(** trainY**(:,**10**)==**1**,:** **);**  TrainX **=** **[** trainX4**;** trainX9 **];**  TrainY **=** **[** zeros**(**size**(**trainX4**,**1**),**1**);** ones**(**size**(**trainX9**,**1**),**1**)** **];**  TestX **=** **[** testX4**;** testX9 **];**  TestY **=** **[** zeros**(**size**(**testX4**,**1**),**1**);** ones**(**size**(**testX9**,**1**),**1**)** **];**  %logistic Regression  tic  nFeature **=** im\_side **^**2**;**  w0 **=** rand**(**nFeature**+**1**,**1**);**  LogisticW **=** logisticRegressionWeights**(** TrainX**,** TrainY**,**w0**,**1000**,**0.1 **);**  LogisticRes **=** logisticRegressionClassify**(** TestX**,**LogisticW **);**  LogisticErr **=** sum**(** abs**(**LogisticRes **-** TestY**)** **);**  LogisticPercent **=** LogisticErr **/**size**(**TestX**,**1**);**  disp**(**string**(**'Worry:'**)+**string**(**LogisticPercent**));**  sum**(**LogisticRes**>**TestY**)**  sum**(**LogisticRes**<**TestY**)**  toc  %Fisher Regression  tic  **[**FisherW**,**FisherW0**]** **=** Fisher**(**TrainX**,**TrainY**);**  FisherRes **=** TestX**\***FisherW**+**FisherW0 **>** 0**;**  FisherErr **=** sum**(** abs**(**FisherRes**-**TestY**)** **);**  FisherPercent **=** FisherErr**/**size**(**TestX**,**1**);**  toc  %%%--Fisher Function to Calculate the w (with inv(S\_w) )  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %  % Fisher.m  %  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  **function** **[**w**,**w0**]** **=** Fisher**(**xT**,**yT**)**  % ybar=w'\*x+w0, ybar>0 ==> x<-w1,y=0  %  logic0 **=** **(**yT **==** 0 **);**  logic1 **=** **(**yT **==** 1 **);**  x0 **=** xT**(**logic0**,:);**  x1 **=** xT**(**logic1**,:);**  m0 **=** mean**(**x0**);**  m1 **=** mean**(**x1**);**  nFeature **=** size**(**m0**',**1**);**  Sw **=** zeros**(**nFeature**,**nFeature**);**  **for** m **=** 1**:**size**(**x0**,**1**)**  temp **=** x0**(**m**,:)** **-**m0**;**  Sw **=** Sw**+**temp**'** **\*** temp**;**  **end**  **for** m **=** 1**:**size**(**x1**,**1**)**  temp **=** x1**(**m**,:)-**m1**;**  Sw **=** Sw**+**temp**'** **\*** temp**;**  **end**  w **=** Sw**\(**m0**-**m1**)';**  w **=** w**/**sqrt**(** sum**(**w **.^**2**)** **);**  w0 **=** **-**1**/**2**\*(**m0**+**m1**)\***w**;**  **if** m0**\***w**+**w0 **>**0  w **=** **-**w**;**  w0 **=** **-**w0**;**  **end**  **end**  %%%--Logistic Regression Packets---  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %  % logisticRegressionWeights.m  %  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  **function** **[**w**]** **=** logisticRegressionWeights**(** XTrain**,** yTrain**,** w0**,** maxIter**,** learningRate**)**  **[**nSamples**,** nFeature**]** **=** size**(**XTrain**);**  w **=** w0**;**  precost **=** 0**;**  **for** j **=** 1**:**maxIter  temp **=** zeros**(**nFeature **+** 1**,**1**);**  Xreal **=** **[**ones**(**nSamples**,**1**)** XTrain**];**  %%Change to Matrix  % for k = 1:nSamples  % temp = temp + (sigmoid( Xreal(k,:)\* w) - yTrain(k)) \* Xreal(k,:)';  % end  sig **=** sigmoid**(** Xreal**\***w **)** **-** yTrain**;**  sig2 **=** repmat**(**sig**,**size**(**1**,**nSamples**+**1**));**  sig2 **=** sig2 **.\*** Xreal**;**  temp **=** sum**(**sig2**)';**  w **=** w **-** learningRate **\*** temp**;**  cost **=** CostFunc**(**XTrain**,** yTrain**,** w**);**  **if** j**~=**0 **&&** abs**(**cost **-** precost**)** **/** cost **<=** 0.0001  **break;**  **end**  precost **=** cost**;**  **end**  **end**  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %  % logisticRegressionClassify.m  %  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  **function** **[** res **]** **=** logisticRegressionClassify**(** XTest**,** w **)**  nTest **=** size**(**XTest**,**1**);**  res **=** zeros**(**nTest**,**1**);**  res **=** sigmoid**(** **[**ones**(**nTest**,**1**)** XTest**]** **\***w **)** **>=** 0.5**;**  res **=** double**(**res**);**  % for i = 1:nTest  % sigm = sigmoid([1.0 XTest(i,:)] \* w);  % if sigm >= 0.5  % res(i) = 1;  % else  % res(i) = 0;  % end  % end  **end**  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %  % sigmoid.m  %  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %Change the function to Matrix Calculate  % ChengjunJia 2017-4-2  **function** **[** output **]** **=** sigmoid**(** input **)**  %output = tanh(input);  output **=** ones**(**size**(**input**))** **./** **(**1 **+** exp**(-** input**));**  **end**  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  %  % CostFunc.m  %  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  **function** **[** J **]** **=** CostFunc**(** XTrain**,** yTrain**,** w **)**  **[**nSamples**,** nFeature**]** **=** size**(**XTrain**);**  temp **=** 0.0**;**  logic1 **=** **(**yTrain **==** 1**);**  hXtemp **=** sigmoid**(** **[**ones**(**nSamples**,**1**)** XTrain**]** **\***w**);**  hx1 **=** hXtemp**(**logic1**);**  hx2 **=** hXtemp**(~**logic1**);**  temp **=** sum**(**log**(**hx1**))+**sum**(**log**(**1**-**hx2**));**  %%Change to the matrix  % for m = 1:nSamples  % hx = sigmoid([1.0 XTrain(m,:)] \* w);  % if yTrain(m) == 1  % temp = temp + log(hx);  % else  % temp = temp + log(1 - hx);  % end  % end  J **=** temp **/** **(-**nSamples**);**  **end** |

1. Logistics把**原来的部分运算**改为向量运算，以提高效率 [↑](#footnote-ref-1)
2. 这里的错误率是指：错误的判断结果/总的判断数目；即：4误判9/4的总数目 [↑](#footnote-ref-2)
3. 先验概率之比取1——认为4和9的出现概率等价 [↑](#footnote-ref-3)
4. 需要考虑到神经网络训练的结果是针对所有类别的结果，可以重新设置神经网络进行训练（本文没有进行这方面的工作） [↑](#footnote-ref-4)