

成 绩

MNIST手写字体分类

实验报告

院 （系） 名称 自动化科学与电气工程学院

专 业 名 称 自动化

学 生 学 号 15031184

学 生 姓 名 李思奇

2018年6月14日

# **摘要**

MNIST数据集来自美国国家标准与技术研究所，**National Institute of Standards and Technology (NIST)**。训练集由来自250个不同人手写的数字构成，其中50%是高中学生，50%来自人口普查局的工作人员。测试集也是同样比例的手写数字数据。本文中使用人工神经网络以及KNN算法两种方法实现多分类问题。

# 一、人工神经网络(ANN)算法

## 1.1 简介

人工神经网络(ANN)的学习模式是一种受生物神经系统启发的方法，它在现实世界中执行异常复杂的计算，而不依赖于显式的定量操作。这项技术最初的灵感来自于生物大脑中由神经元及其突触形成的生物电网络。在神经网络模型中，一些简单的节点(称为“神经元”或“单元”)连接在一起形成一个节点网络，因此称为“神经网络”。

每个节点都有一组输入线，它们类似于生物神经元中的输入突触。每个节点也有一个“激活函数”，它告诉节点何时激活，类似于生物神经元。通常，神经网络至少有三层：输入层、隐藏层和输出层。输入层不进行处理，它只是将数据向量输入神经网络。数据经过输入层后输入隐藏层。隐藏层依次向输出层输入。神经网络中，实际的数据处理发生在隐藏层和输出层的节点上。

## 1.2利用BP算法训练神经网络

### 1.2.1 模型及算法

MNIST数据集是一个多分类数据集，包含训练样本60000个，测试样本10000个，特征维度为784，输出类别数为10。在本次试验中，考虑到运算效率等因素，仅采用10%的数据，即6000个训练样本和1000个测试样本。构造一个3层神经网络，输入层神经元数为784，隐含层神经元数为100，输出层神经元数为10，神经网络结构简图如图2.1所示。

1

1

x1

x2

y1

y10

Input layer

Hidden layer

Output layer

a1

a2

a3

a100

……

……

x784

……

**图2.1 神经网络结构简图**

设输入为：

 (2-1)

输出为：

 (2-2)

记样本数为m，为样本x通过神经网络前向传播得到的输出。为从第j层到第j+1层的权重矩阵。

则有：





其中，为Sigmoid函数。

定义代价函数为：



其中，后一项相当于在logistic regression中的正则化项，目的是防止过拟合的现象出现。

训练神经网络的目标便是最小化损耗函数。我们使用误差反向传播算法(Back Propagation)来实现。

计算每层每个结点的误差：

 (2-3)

则令：



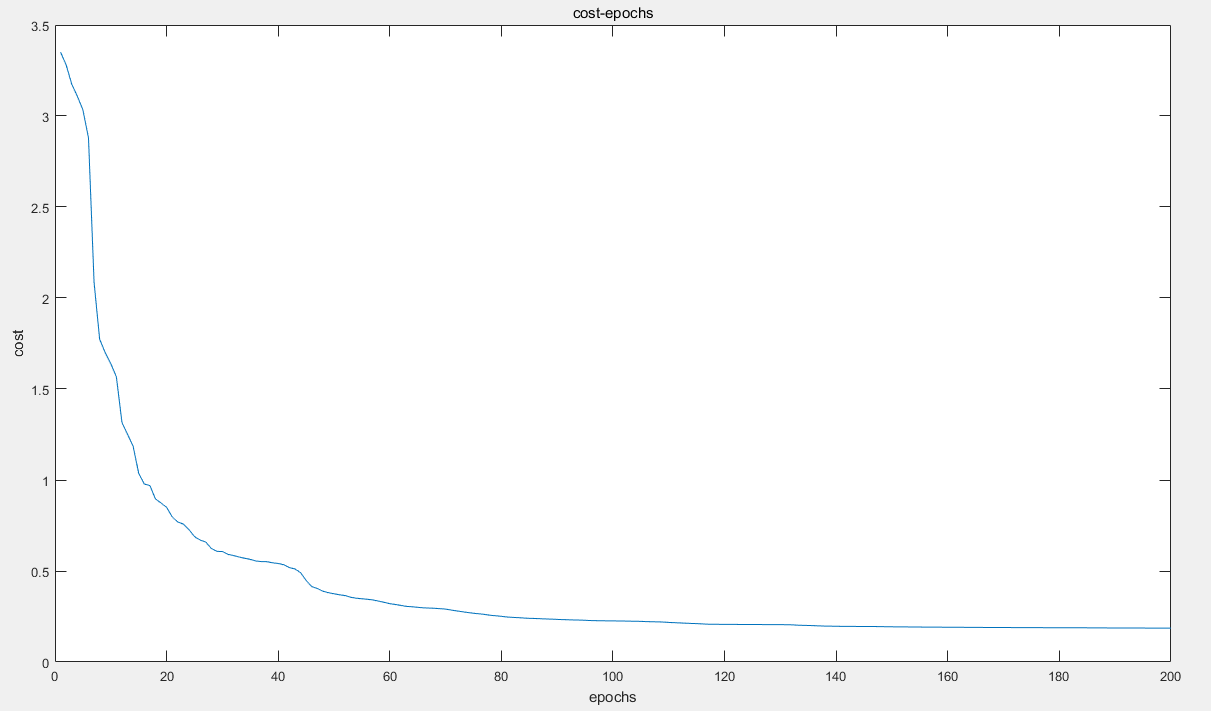
有：

 (2-4)

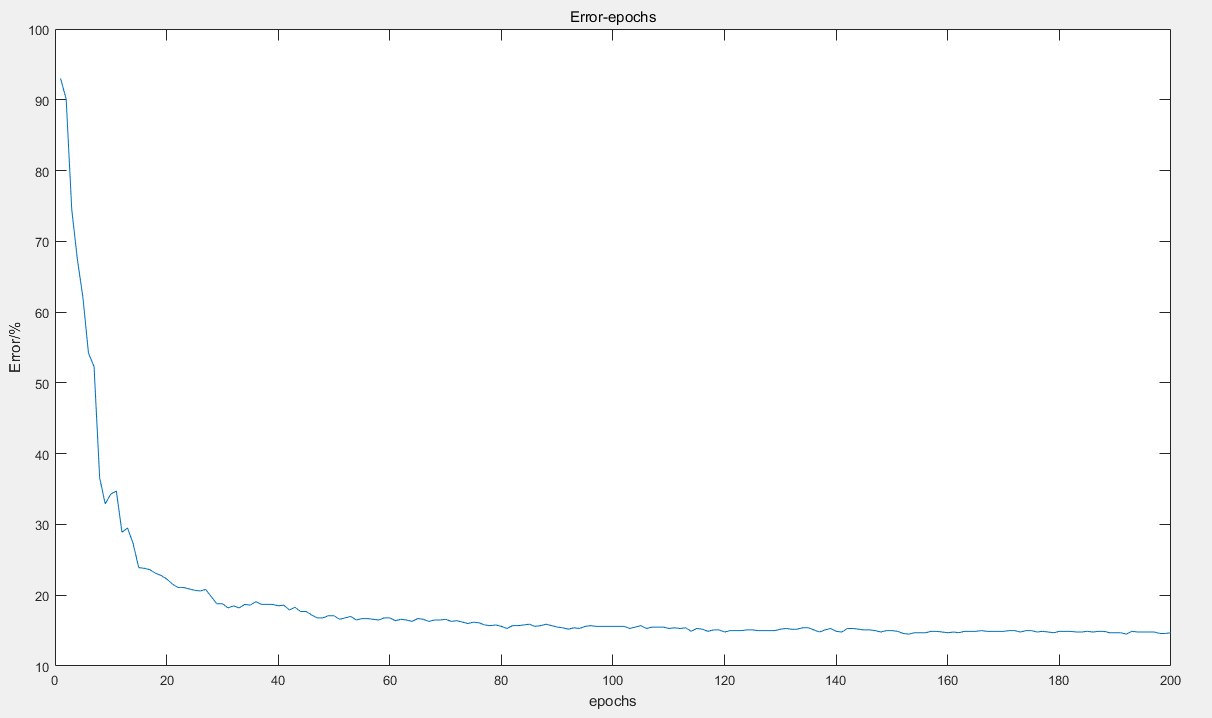
利用式(2-4)得到的梯度值，利用梯度下降算法，可以得到优化的权重矩阵。

### 1.2.2 训练结果

对神经网络，使用训练集利用BP算法进行训练。设置正则化系数，训练神经网络。经过200次迭代，做出代价函数与分类错误率关于epoch的曲线如图2.2和图2.3所示。



**图2.2 代价函数曲线**



**图2.3 错误率曲线**

最终分类结果如图2.4所示。可以看到，利用神经网络分类结果错误率为14.7%，错误率较高。



**图2.4 分类结果**

# 二、KNN算法

## 2.1 简介

KNN (K-Nearest Neighbor) 算法即K最邻近算法，是实现分类器中比较简单易懂的一种分类算法。该算法的思想是：一个样本与数据集中的K个样本最相似，如果这K个样本中的大多数属于某一个类别，则该样本也属于这个类别。K临近之所以简单是因为它比较符合人们直观感受，即人们在对事物进行分类的时候，最容易想到的就是离哪一类最近就属于哪一类。K-NN算法就是基于欧几里得距离推断事物类别的一种实现方法。

该算法涉及3个主要因素：实例集、距离的衡量、K的大小。

KNN算法描述如下：

Step1：初始化训练集和类别；

Step2：计算测试集样本与训练集样本的欧氏距离；

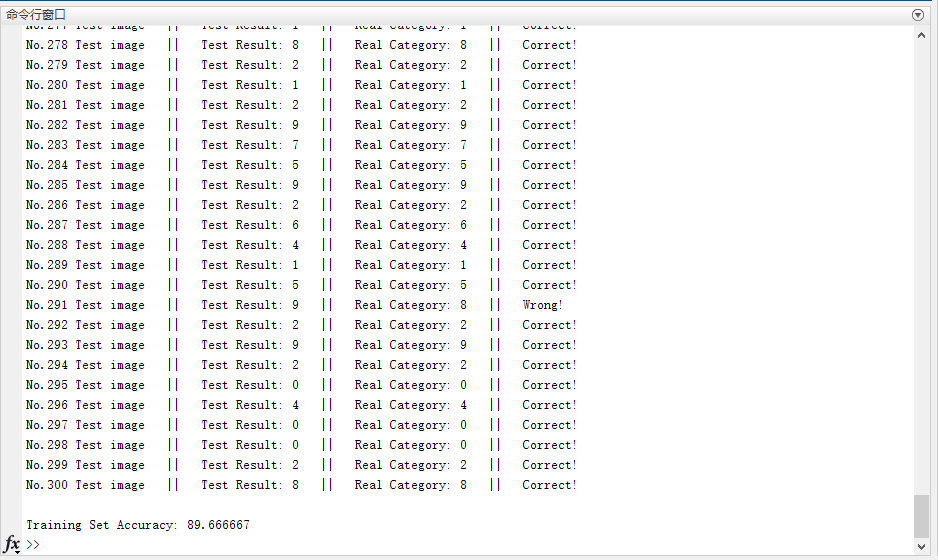
Step3：选取欧式距离最小的前K个训练样本，统计其在各类别中的频率；

Step4：返回频率最大的类别，即测试集样本属于该类别。

## 2.2 分类结果

考虑到效率原因，选取60000个训练集中的20000个，选取10000个测试集中的300个进行测试。

取K=100进行分类测试，测试结果如图2.1所示.



**图2.1 KNN算法分类结果**

可见，使用KNN算法分类正确率为89.67%，略高于神经网络算法。

# 三、总结

本文中，使用人工神经网络及KNN算法对MNIST手写字体数据库进行分类，均有着不错的效果。下面简单分析这两种算法的优点与缺点。

## 3.1 人工神经网络优缺点

### 3.1.1 ANN优点

(1) 有非线性映射能力

BP神经网络实质上实现了一个从输入到输出的映射功能，数学理论证明三层的神经网络就能够以任意精度逼近任何非线性连续函数。这使得其特别适合于求解内部机制复杂的问题。

(2) 自学习和自适应能力

BP神经网络在训练时，能够通过学习自动提取输出、输出数据间的“隐含规则”，并自适应的将学习内容记忆于网络的权值中。即BP神经网络具有高度自学习和自适应的能力。

(3)容错性好

BP神经网络在其局部的或者部分的神经元受到破坏后对全局的训练结果不会造成很大的影响。

### 3.1.2 ANN缺点

(1) 易陷入局部极小化问题

从数学的角度来看，传统的BP神经网络局部搜索优化方法，它是一个复杂的非线性问题解决，网络权值是通过局部改进方向逐渐调整，所以可以使算法陷入局部极值，收敛到局部最小值点的权重，从而导致网络训练的失败。BP神经网络对初始网络的重量非常敏感。当用不同的权重初始化网络时，它趋向于收敛于不同的局部极小值。

(2) 收敛速度慢

由于BP神经网络算法本质上是梯度下降法，使得BP算法效率低下。由于优化的目标函数是非常复杂的，它一定是在神经元输出接近于0或1的条件下，在这些区域中，权重误差变化很小，训练过程几乎停止。

(3) 神经网络结构不定

到目前为止，对BP神经网络的结构选择没有统一和完整的优化理论，只能通过经验来选择。网络结构的选择太大，训练效率不高，可能出现超拟现象，导致网络性能较低。如果选择太小，网络可能不会收敛。因此，如何选择网络结构是一个重要的问题。

## 3.2 KNN优缺点

### 3.2.1 KNN优点

①简单，易于理解，易于实现，无需参数估计，无需训练；

②精度高，鲁棒性强，对异常值不敏感（个别噪音数据对结果的影响不是很大）；

③特别适合于多分类问题。

### 3.2.2 KNN缺点

①对测试样本分类时的计算量大，因为对每一个待分类的样本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点。因此在测试时很费时间。

②可解释性差，无法给出决策树那样的明确规则；

③当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数。这会导致错误分类。

附录：Matlab程序

1. 人工神经网络

%==============================主程序===============================%

clear;clc;

data\_train = loadMNISTImages('train-images.idx3-ubyte');

labels\_train = loadMNISTLabels('train-labels.idx1-ubyte');

data\_test = loadMNISTImages('t10k-images.idx3-ubyte');

labels\_test = loadMNISTLabels('t10k-labels.idx1-ubyte');

%%读取数据并截取前6000个训练集，前1000个测试集

data\_train = data\_train(:,1:6000);

labels\_train = labels\_train(1:6000);

data\_test = data\_test(:,1:1000);

labels\_test = labels\_test(1:1000);

X=data\_train';

m = size(X, 1);

y = labels\_train;

%%设定神经网络结构

input\_layer\_size = 784; hidden\_layer\_size = 100; num\_labels = 10;

%%初始化参数

initial\_Theta1 = randInitializeWeights(input\_layer\_size, hidden\_layer\_size);

initial\_Theta2 = randInitializeWeights(hidden\_layer\_size, num\_labels);

initial\_nn\_params = [initial\_Theta1(:) ; initial\_Theta2(:)];

%%梯度下降迭代

lambda = 1;

ep = 200;

options = optimset('MaxIter', ep);

costFunction = @(p) nnCostFunction(p,input\_layer\_size,hidden\_layer\_size,num\_labels, X, y, lambda);

[nn\_params, cost, Acc] = fmincg(costFunction, initial\_nn\_params, options);

%%绘图

figure(1);

plot(1:ep,cost);

title('cost-epochs');xlabel('epochs');ylabel('cost');

figure(2);

plot(1:ep,100 - Acc);

title('Error-epochs');xlabel('epochs');ylabel('Error/%');

%%计算错误率

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)), hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):end), num\_labels, (hidden\_layer\_size + 1));

pred = predict(Theta1, Theta2, data\_test');

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == labels\_test)) \* 100);

%=============================初始化参数=============================%

function W = randInitializeWeights(L\_in, L\_out)

W = zeros(L\_out, 1 + L\_in);

epsilon\_init = 0.12;

%%在正负epsilon\_init之间生成随机数

W = rand(L\_out,1 + L\_in) \* 2 \* epsilon\_init - epsilon\_init;

end

%============================Sigmoid函数=============================%

function g = sigmoid(z)

g = 1.0 ./ (1.0 + exp(-z));

end

%============================Sigmoid导数=============================%

function g = sigmoidGradient(z)

g = zeros(size(z));

g= sigmoid(z) .\* (1-sigmoid(z));

end

%===========================代价函数及梯度===========================%

function [J grad] = nnCostFunction(nn\_params,input\_layer\_size,hidden\_layer\_size,num\_labels,X, y, lambda)

%%整合参数、初始化

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)),hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):end), num\_labels, (hidden\_layer\_size + 1));

m = size(X, 1);

J1 = 0; J2 = 0;

Theta1\_grad = zeros(size(Theta1));

Theta2\_grad = zeros(size(Theta2));

X = [ones(m,1),X];

Y = zeros(m,num\_labels);

for i=1:m

if y(i)==0 Y(i,10)=1;

else Y(i,y(i))=1;

end

end

Y = Y';

%%正向传播计算输出

Y = Y';

z2 = Theta1 \* X';

a2 = sigmoid(z2);

a2 = [ones(1,m);a2];

z3 = Theta2 \* a2;

hx = sigmoid(z3);

%%计算代价

for i=1:m

for k=1:num\_labels

J1 = J1 - Y(k,i)\*log(hx(k,i))-(1-Y(k,i))\*log(1-hx(k,i));

end

end

J1 = J1/m;

for i=1:hidden\_layer\_size

for k=1:input\_layer\_size

J2 = J2 + Theta1(i,k+1)^2;

end

end

for i=1:num\_labels

for k=1:hidden\_layer\_size

J2 = J2 + Theta2(i,k+1)^2;

end

end

J2 = J2 \* lambda / 2 /m;

J = J1 + J2;

%%计算梯度

delta3 = hx - Y;

delta2 = Theta2' \* delta3 ;

delta2 = delta2(2:hidden\_layer\_size+1,:);

delta2 = delta2 .\* sigmoidGradient(z2);

dta1 = zeros(hidden\_layer\_size,input\_layer\_size+1);

dta2 = zeros(num\_labels,hidden\_layer\_size+1);

dta1 = dta1 + delta2 \* X;

dta2 = dta2 + delta3 \* a2';

t1 = lambda\*[zeros(hidden\_layer\_size,1),Theta1(:,2:input\_layer\_size+1)];

t2 = lambda\*[zeros(num\_labels,1),Theta2(:,2:hidden\_layer\_size+1)];

Theta1\_grad = dta1/m+t1/m;

Theta2\_grad = dta2/m+t2/m;

grad = [Theta1\_grad(:) ; Theta2\_grad(:)];

end

%=============================计算正确率=============================%

function p = predict(Theta1, Theta2, X)

m = size(X, 1);

num\_labels = size(Theta2, 1);

p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X] \* Theta1');

h2 = sigmoid([ones(m, 1) h1] \* Theta2');

[dummy, p] = max(h2, [], 2);

end