《深度学习引论》课程报告

**课题名称： 《深度学习引论》期末结课设计**

课题负责人名（学号）： 马晓晨 2019141460009

同组成员名单（角色）：

无

指导教师： 章毅

**评阅成绩：**

评阅意见：

提交报告时间：2020年12月5日

[《深度学习引论》课程报告 0](#_Toc57986765)

[**1** 课程设计问题概述 4](#_Toc57986766)

[**1.1** The Street View House Numbers (SVHN) 数据集[1] 4](#_Toc57986767)

[**1.2** 数据集格式 4](#_Toc57986768)

[**1.3** 数据集目标任务 5](#_Toc57986769)

[**2 解决方案设计** 5](#_Toc57986770)

[**2.1 基本思路** 5](#_Toc57986771)

[**2.2 设计的难点** 6](#_Toc57986772)

[**3** 数据预处理 7](#_Toc57986773)

[**3.1** 彩色图片转换为灰度图片并归一化 8](#_Toc57986774)

[3.1.1 转换思路 8](#_Toc57986775)

[3.1.2 实现过程 8](#_Toc57986776)

[**3.2** 整理矩阵为向量 9](#_Toc57986777)

[3.2.1 处理过程 9](#_Toc57986778)

[**3.3** 处理结果 10](#_Toc57986779)

[**4** 神经网络结构 10](#_Toc57986780)

[**4.1** 基本模型 10](#_Toc57986781)

[**4.2** 神经网络框架 11](#_Toc57986782)

[4.2.1 层数 11](#_Toc57986783)

[4.2.2 每层神经元个数 11](#_Toc57986784)

[**4.3** 激活函数选取 11](#_Toc57986785)

[4.3.1 Simoid函数[4] 11](#_Toc57986786)

[4.3.2 线性整流函数（ReLU）[4] 12](#_Toc57986787)

[**5** 训练过程 12](#_Toc57986788)

[**5.1** 激活函数为Sigmoid函数的全连接神经网络训练 12](#_Toc57986789)

[5.1.1 基本参数 12](#_Toc57986790)

[5.1.2 训练情况 13](#_Toc57986791)

[5.1.3 训练结果 15](#_Toc57986792)

[**5.2** 主要激活函数为线性整流函数（ReLU）配合Sigmoid函数输出的神经网络模型 16](#_Toc57986793)

[5.2.1 基本参数 16](#_Toc57986794)

[5.2.2 训练情况 17](#_Toc57986795)

[5.2.3 训练结果 20](#_Toc57986796)

[**6** 实验结果 20](#_Toc57986797)

[**7** 小结 21](#_Toc57986798)

[**8** 代码 21](#_Toc57986799)

[**8.1** ReLU训练用trainmodle.m代码 21](#_Toc57986800)

[**8.2** ReLU神经网络函数 24](#_Toc57986801)

[8.2.1 fc.m (ReLU的正向传播) 24](#_Toc57986802)

[8.2.2 fc\_sig.m （Sigmoid的正向传播） 24](#_Toc57986803)

[8.2.3 bc.m (ReLU的反向传播) 24](#_Toc57986804)

[8.2.4 bc\_sig.m （Sigmoid的反向传播） 24](#_Toc57986805)

[8.2.5 cost.m （cost function） 24](#_Toc57986806)

[**9** 参考文献 25](#_Toc57986807)

**深度学习引论课程期末项目**

计算机科学与技术 专业

**学生** 马晓晨 **指导老师** 章毅

**[摘要]** 随着深度学习课程的深入了解，需要将本学期学习的神经网络知识运用到实践操作，以确保自己掌握训练集与测试集的预处理，神经网络的设计，BP算法反向传播的应用，调参技巧等等神经网络的知识，证明自己基本掌握深度学习的基础原理。利用了SVHN谷歌街景门牌号码作为数据集和测试集进行训练，分别采用sigmoid与RELU作为激活函数进行设计，最终实现正确率达到80%以上的神经网络模型。

**关键词**： 神经网络 深度学习 反向传播

# 课程设计问题概述

## The Street View House Numbers (SVHN) 数据集[1]

SVHN（Street View House Number）Dateset，谷歌街景门牌号码数据集（<http://ufldl.stanford.edu/housenumbers/>），用于训练机器学习能力以及识别算法。作为从谷歌街景中裁切的数字图像，它与手写体数字的MNIST数据集非常类似，然而作为来自实景照片的数字图片集，它拥有更多的数据量，以及更高的识别难度。

有10种可能的情况，数字‘1’拥有标签1，数字‘2’对应标签2，之后每个数字一一对应，除了‘0’对应的是标签10。

训练集拥有73257个数据，测试集有26032个数据。并且拥有一个有531131个更容易识别的样本的额外集。

## 数据集格式

数据集有如下两种格式：

有方框标识各个数字位置的原始街景大图

和MNIST数据集类似的32×32的切分好的方形数字图片（但有的图片旁边会有相邻的数字入镜）

本次课程设计只利用格式②中的数据集，故仅介绍格式②中数据。

数据②中有3个文件，分别是

train\_32x32.mat （73257张图片）

test\_32x32.mat （26032张图片）

extra\_32x32.mat （531131张图片）

它们都是基于Matlab的裁切好的图片矩阵存成的文件。

每个数字都被按照合适的缩放比例裁切到正方形，然后缩放为32×32像素RGB图片的格式，所以调整图片素材为32×32的大小并不会引起图片长宽比例失真。然而，这种预处理会将原数字旁边的一些数字引入到我们“感兴趣”的数字侧面，可能会影响识别。

三个\*.mat文件加载时都会创建两个变量：X是包含图像的4-D Matrix（对于train\_32x32.mat，其中的X是32x32x3x73257的矩阵），y是包含标签的向量，读取时通过X(:,:,:,i)获得每一张32×32的图像，其中RGB数据被存储于0-255的uint8数据单元内，每一张图像文件可以和y(i)所存储的标签进行一一匹配。

## 数据集目标任务

本数据集最终希望实现的目标为，通过某种方式，建立从街景门牌号图片信息到其实际数值的映射，即对于所有图片生成一个10元素分类系统，以简单实现图像识别中的OCR（光学字符识别）技术，可以通过电脑自动识别每一个门牌的数字所表示的数值。

# **解决方案设计**

## **基本思路**

依据本课程中学习的MNIST[2]手写体数字识别技术（Lab4）的解决方案，采用神经网络实现对于训练集的分类和识别，并投入测试集进行模型的效果检验，取得了训练集正确率高达98%以上，测试集正确率高达95%以上的效果，结果见图 2‑1。验证了其可行性，并基本掌握了利用神经网络识别手写体数字的技术。

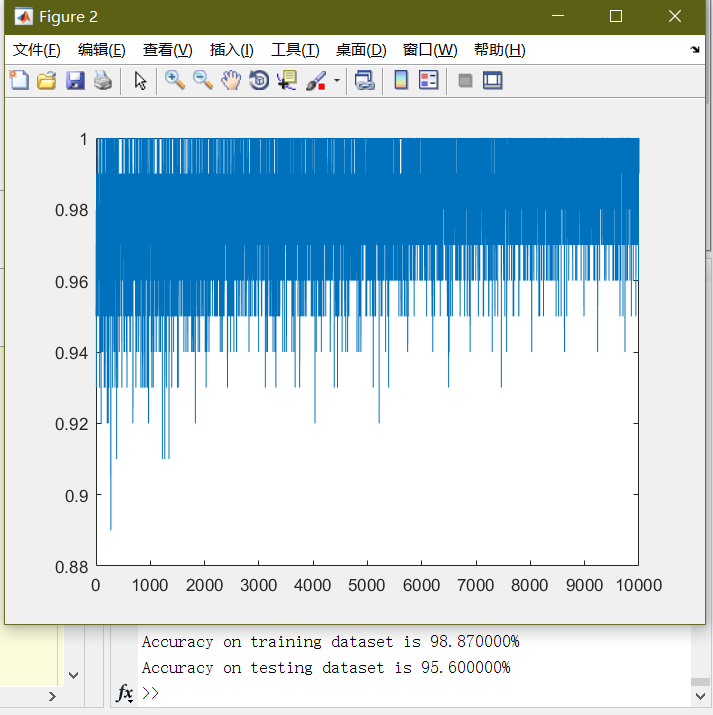


图 2‑1 Lab4中对于MNIST的训练效果，图例为101~200次训练的ACC

那对于本次高度相似的数字识别任务，我们依旧采用神经网络完成任务，考虑到课程要求不使用框架，同样需要像Lab4一样，自己构建神经网络模型，并调整参数进行训练，实现最后的分类、识别功能。

## **设计的难点**

考虑到本次图片中有大量的干扰信息，并且色彩方面更加复杂。相比于MNIST高对比度，黑白分明，特征清晰的灰度图像数据，见图 2‑2；SVHN图片在图片明度、色彩、字体特征杀好难过更加杂乱，并且存在相邻干扰数字，理论上会大幅影响模型的分类难度，见图 2‑3。所以预期的正确率应该会想比MNIST有相当程度的降低，要考虑如何最大程度的整理数据集，提前整理特征解决潜在的问题。

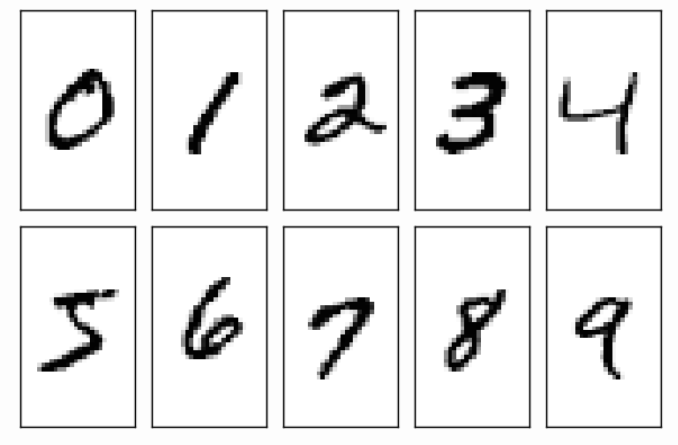


图 2‑2 MNIST数据集图片样例



图 2‑3 SVHN数据集图片样例

# 数据预处理

所有预处理过程由代码中的read.m脚本实现，将格式②中的训练集和测试集的信息，按照需求转化成的X\_train，X\_test，trainlables，test\_label矩阵（向量），并分别保存于train\_1024.mat和test\_1024.mat文件中。文件见图 3‑1

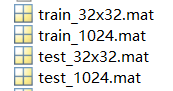


图 3‑1 原始数据文件以及生成的处理后的数据文件

在训练时，直接读取test\_1024.mat的变量即可，测试时，直接读取test\_1024.mat中的变量即可

## 彩色图片转换为灰度图片并归一化

### 转换思路

考虑到对于每一张谷歌街景数字的图片数据，仅识别其中央的数字的具体数值，只需要保留其色彩大幅变化轮廓的形状特征即可完成识别，在这样的过程中，图像的色彩数据基本不会影响对数字的分类，为了减少无用特征对于网络的干扰，提高训练速度与准确度，选择将彩色RGB图片转化为灰度图片。

考虑到神经网络在处理时乘以权值矩阵时会产生小数，为保留其精确度需要转换为double类型，并且为了防止采用的sigmoid激活函数输入绝对值过大引起神经元输出饱和的现象，需要将数据进行归一化，提高神经网络的收敛性。

### 实现过程

通过在matlab中调用rgb2gray函数，读取文件中的X向量（图片4-D矩阵）以后，可以将X中的32x32x3的uint8 0~255 RGB图片数据矩阵转化为数值为0~255的uint8 32x32的灰度图像矩阵，然后对该图像矩阵进行归一化处理，将其整体除以255，即得到了数值分布在[0,1]闭区间内的灰度图像矩阵32x32x73257（以train为例）。对于测试集的处理过程同上。此时得到32x32的灰度图像，结果见图 3‑2。



图 3‑2 灰度化归一化后的SVHN图片样本（前100个）

可以看到，图像中的数字特征仍然保留，可以清晰辨认每一个数字，说明这样的处理方式是有潜力的，不会影响到图片本身的信息质量，也减轻了网络的运算负担和色彩带来的噪音影响。

## 整理矩阵为向量

### 处理过程

基于得到的图片数据，考虑到要输入神经网络进行分析，要将32x32的数据整理成1024维向量，这个过程采用matlab中的reshape函数进行实现，并依次命名为X\_train（1024x73257）和X\_test（1024x26032），分别对应训练集数据和，与此同时要将训练集的标签y(73257x1维向量)转化成trainlabels（10x73257的矩阵，每个10维向量只有对应位为1，其余维度均为0），便于神经网络进行归一化的正向传播，映射到对应的数值。并设置成与图像数据的切片维度的方向相同的向量，便于训练到最后一层时直接比较两个向量是否相等即可得到Accuracy信息。

## 处理结果

所有数据均转化为灰度图像整合成的向量的集合，并且保存了对应的标签向量和size数值，见图 3‑3和图 3‑4

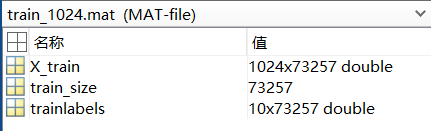


图 3‑3 trian\_1024.mat中的变量

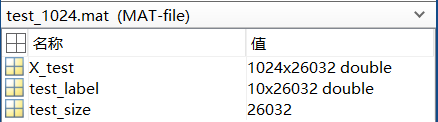


图 3‑4 test\_1024.mat中的变量

由此基本完成了所有数据的预处理，经过验证保留了图片的基本特征，并进行了归一化便于利用sigmoid函数进行训练，转换为向量集合以后数据仍然符合预期，符合神经网络的输入要求，预处理完成！

# 神经网络结构

## 基本模型

采用最简单的神经网络模型，输入为1024维向量，输出为10维向量，没有中间输入神经元或循环输入神经元等设计，基于MATLAB进行模型构建。

## 神经网络框架

### 层数

选择使用5层神经元来构建神经网络模型，不会使模型太浅难以有效拟合模型，也不会使整体层数过深导致运算效率过低。

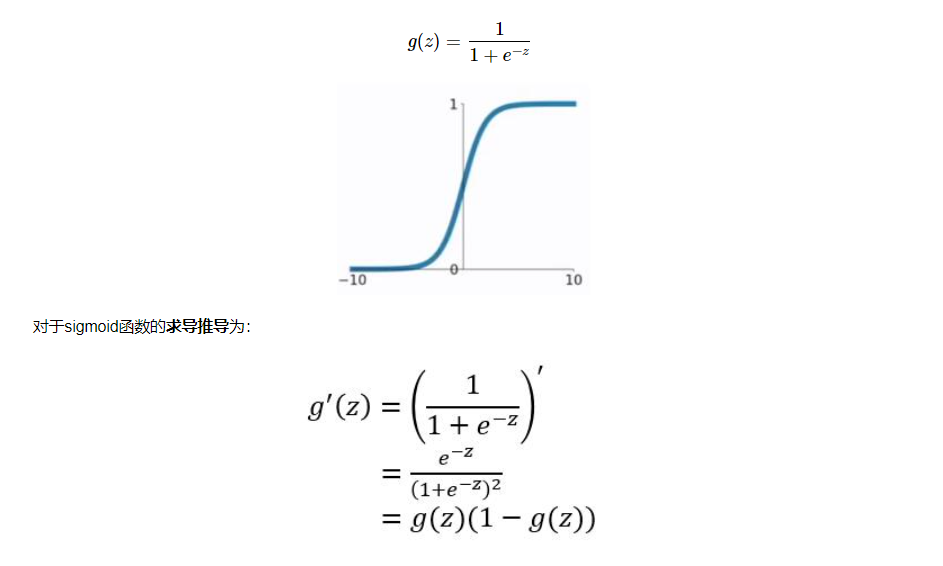
### 每层神经元个数

经过不断调整参数进行设计，观察不同神经元个数对于网络的拟合效率影响，在首层输入1024个数据，并最后分类到10个数据的情况下，经过尝试，每一层的神经元个数选择上最后采用了1024->512->256->64->10的5层设计，基本上可以较为有效的完成目标任务。

## 激活函数选取

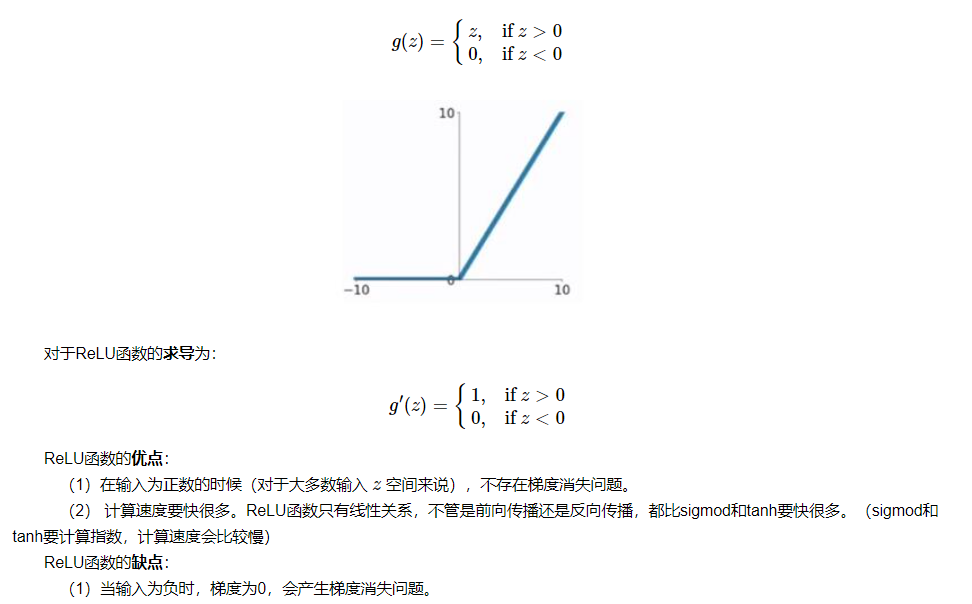
本次实验尝试了两种激活函数进行训练，分别是sigmoid函数与线性整流函数，其详细优势劣势分析均在下一节阐述

### Simoid函数[4]



Sigmoid是第一次MNIST作业中使用的激活函数，已经验证可以用来高效完成手写体数字识别的任务，所以仍然选用该激活函数进行训练。

### 线性整流函数（ReLU）[4]



因为对于图像信息文件，所有的数据都是集中在[0,1]区间内的double类型，所以不用担心ReLU函数负值梯度为0的缺点，又具有比sigmoid函数更高的运算效率，是很有潜力的激活函数。

# 训练过程

## 激活函数为Sigmoid函数的全连接神经网络训练

### 基本参数

对于该网络，初始详细参数如下表

|  |  |
| --- | --- |
| Alpha（学习率） | 0.005 |
| W初始化公式 | w{l} = 0.1 \* randn(layer\_size(l+1,1), sum(layer\_size(l,:))); |
| Train\_size(训练集大小) | 73257 |
| Mini\_batch(取样大小) | 100 |
| Layer\_size（神经元数量） | [1024,512,256,64,10] |
| L（网络层数） | 5 |
| Cost function | MSE |

所有层的激活函数均为sigmoid函数，网络结构如下：

function: **Sigmoid**

The number of neurons: 1024

function: **Sigmoid**

The number of neurons:512

function: **Sigmoid**

The number of neurons:256

function: **Sigmoid**

The number of neurons: 64

function: **Sigmoid**

The number of neurons: 10

output

intput

### 训练情况

首先的max\_epoch设定为35次进行尝试训练。在开始的第一次输入训练集时，cost\_function有指数级大幅下降的趋势，随后长期稳定在0.45左右震荡，Accuracy也在20%左右震荡，如图 5‑1所示。 第一次输入训练集的cost\_function。在前20次输入训练集基本都稳定在该水平，没有呈现进一步收敛的趋势。

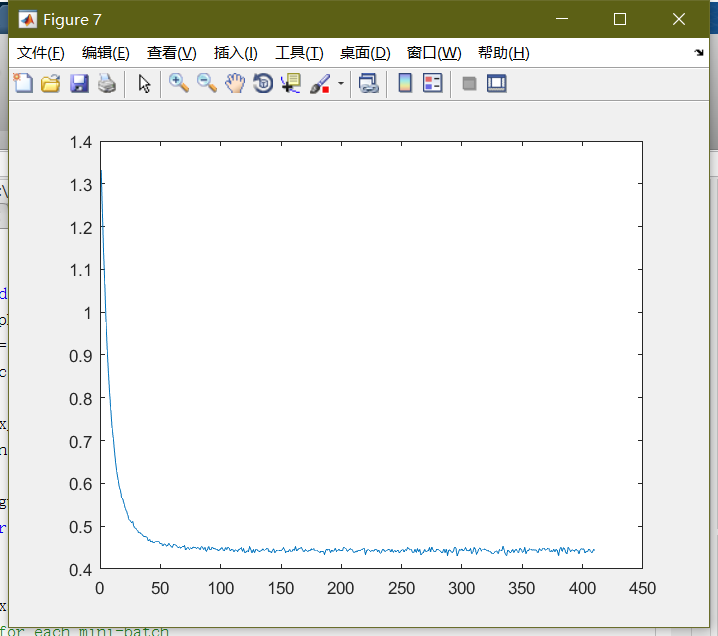


图 5‑1 第一次输入训练集的cost\_function

在一度认为模型设计有误，准备停止训练之前，在第31次输入训练集时，cost\_function突然开始下降，重新收敛，如图 5‑2显示过程。

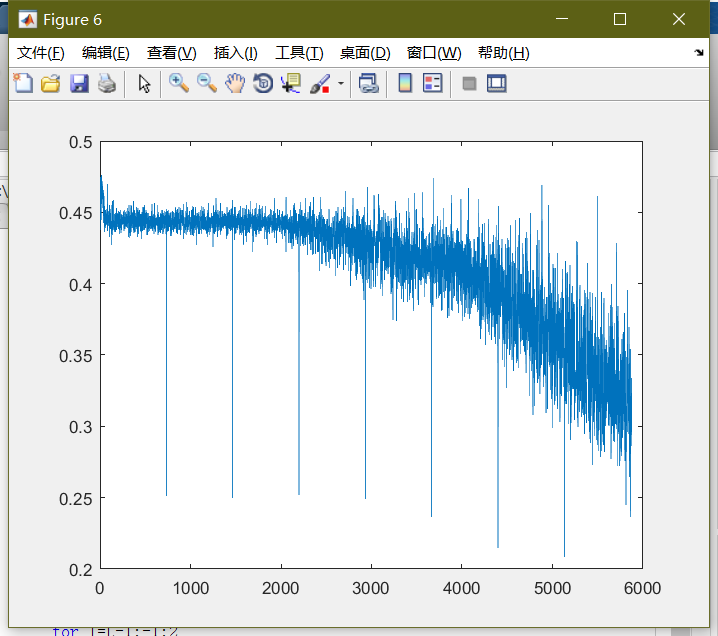


图 5‑2 在第30次输入附近，cost\_funtion开始重新收敛

在此基础上，连续追加了40次输入训练集的训练过程，在总计75次训练时获得了测试集正确率74.11%的训练结果，如图 5‑3所示。验证了整个模型可以得到尚可的分类结果，此时大约花费时间10分钟左右

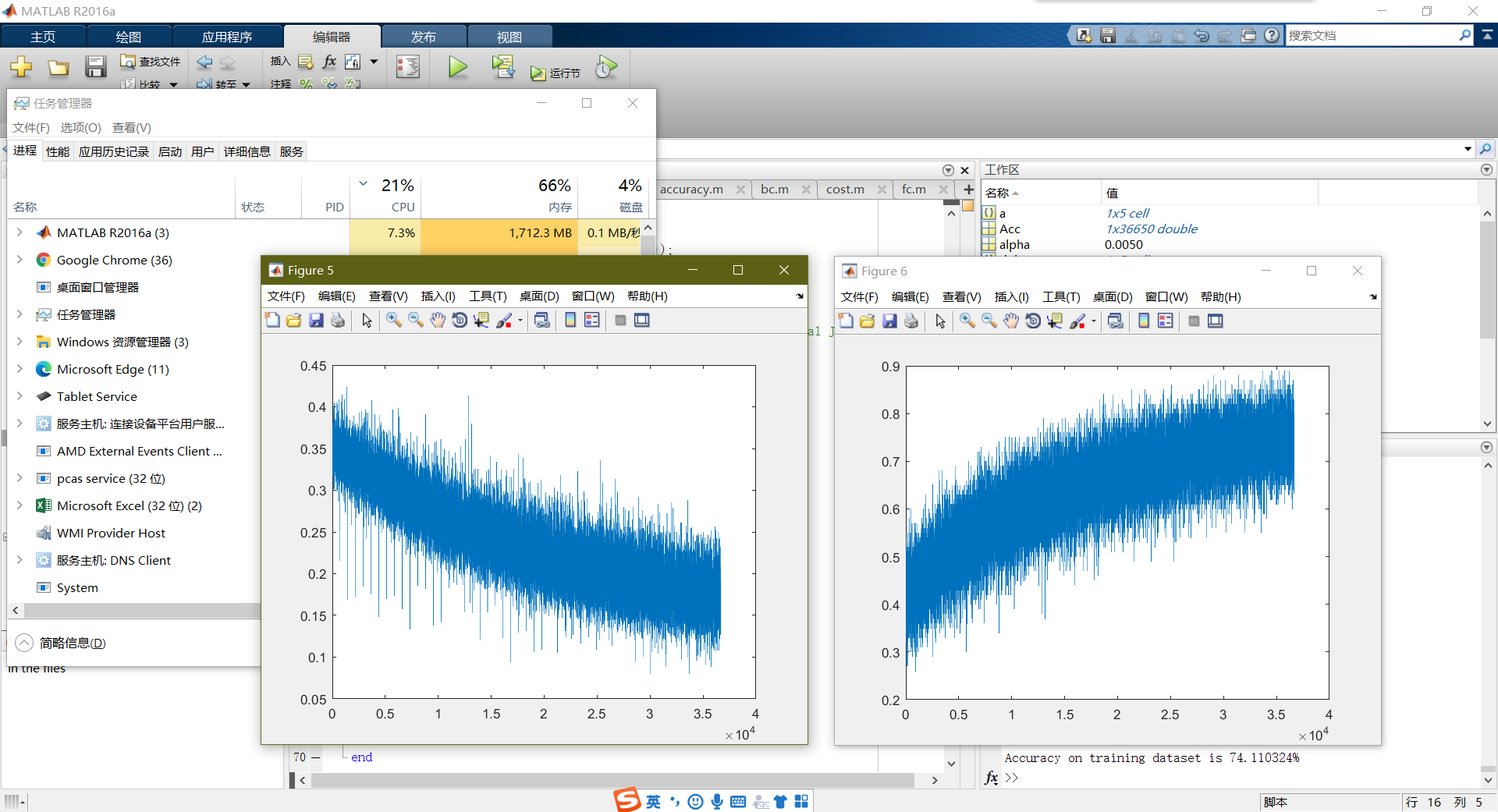


图 5‑3 第75次训练结束的结果，左侧是J，右侧是ACC

在这个基础上，又追加了100次训练，大约花费了半个多小时，随着不断对模型的拟合，对于测试集的正确率有一定的提升。得到的ACC图线如图 5‑4所示。

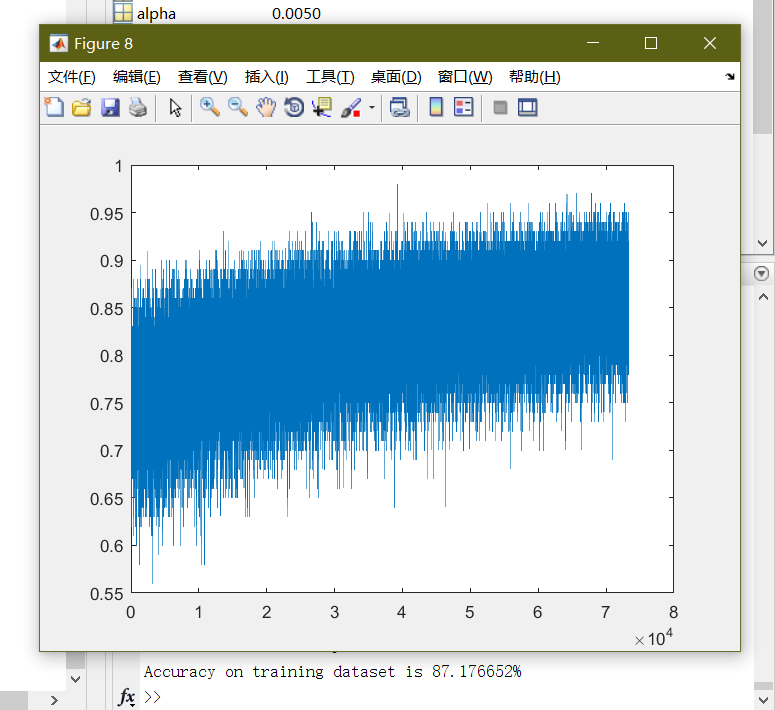


图 5‑4 第75~175次训练时的ACC图线

基本将SVHN识别的神经网络模型的正确率训练到可以接受的范围。

### 训练结果

最终经过训练，花费近1小时的训练时间，最终获得了训练集正确率87.657%，测试集81.918%的谷歌街景门牌识别模型。如图 5‑5。基本可以认为该神经网络模型建立正确。有效的生成了一个正确率有参考意义的SVHN识别模型。



图 5‑5 基于Sigmoid函数的5层全连接模型的最后结果

最后，将矩阵w和layer\_size存储于matlab数据文件model.mat中，如图 5‑6，方便后续研究使用。

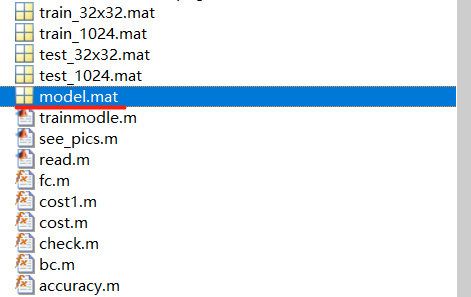


图 5‑6 训练完成后文件夹内的文件一览

## 主要激活函数为线性整流函数（ReLU）配合Sigmoid函数输出的神经网络模型

鉴于Sigmoid函数在学习时效率过低的情况，决定改进神经网络，提升拟合效率和准确度。

### 基本参数

对于该网络，初始详细参数如下表，其中相比于上一个模型，经过调整，逐步降低了学习率以适应ReLU的梯度大幅度降低过程。

|  |  |
| --- | --- |
| Alpha（学习率） | 0.005 到0.001 |
| W初始化公式 | w{l} = 0.1 \* randn(layer\_size(l+1,1), sum(layer\_size(l,:))); |
| Train\_size(训练集大小) | 73257 |
| Mini\_batch(取样大小) | 100 |
| Layer\_size（神经元数量） | [1024,512,256,64,10] |
| L（网络层数） | 5 |
| Cost function | MSE |

其激活函数及网络具体结构如下：

function: **ReLU**

The number of neurons: 1024

function: **ReLU**

The number of neurons:512

function: **ReLU**

The number of neurons:256

function: **ReLU**

The number of neurons: 64

function: **Sigmoid**

The number of neurons: 10

output

intput

在得到该模型之前，尝试了第一层Sigmoid、第一层和最后一层Sigmoid、全ReLU等等情况，最终发现以上模型效果最好，故最终记录该模型。

### 训练情况

该模型的结果是在大量尝试，调试的基础之上实现的，在前期测试中产生了大量只能收敛到正确率20%，无法继续训练的模型，以及随着反向传播，大量w变成负值。梯度消失无法训练的情况。

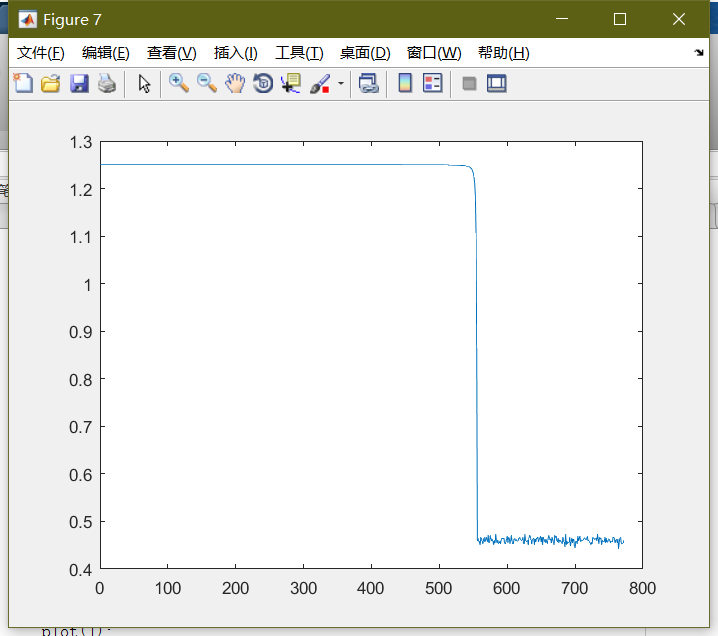


图 5‑7 错误模型之一，前期快速收敛后J只能在0.45左右震荡

随着不断的调试参数，观察不同情况下神经网络的学习状态，对异常的梯度、w中大量为0的种种情况进行分析解决，例如图 5‑8梯度爆炸情况，选择用合适参数初始化矩阵以及调整学习率的方法重新改进。

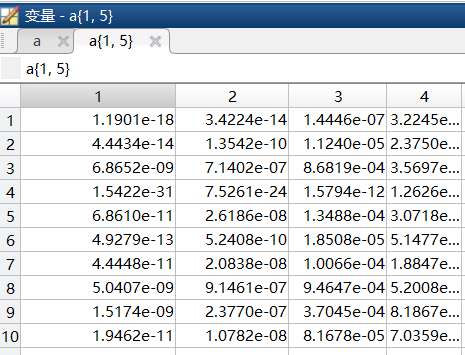


图 5‑8 因为梯度爆炸导致输出层数值异常小的情况

经过不断地变换模型设计，发现在前期经由ReLU实现快速稀疏拟合，最后经由Sigmoid整理输出的模型具有很优秀的性质，最后采用该模型。

在第五次输入训练集时，就拥有了近60%的正确率，用时两分钟不到，训练时截图如图 5‑9所示，

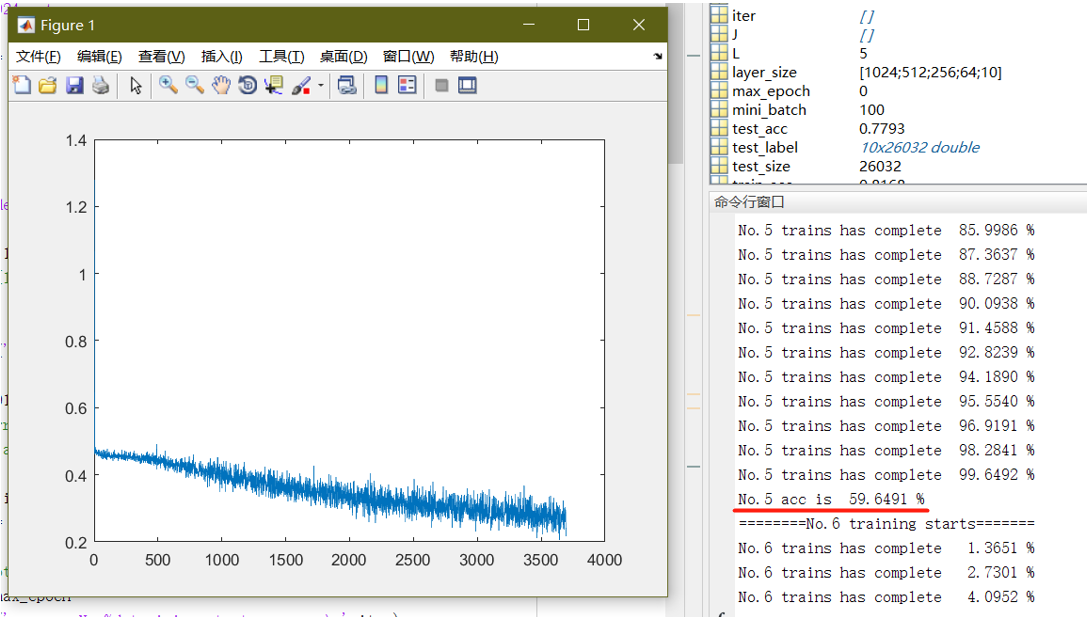


图 5‑9 第五次训练已经达到59.649%的正确率

在第十次结束以后发现cost\_function下降速度放缓，于是减小了学习率α到0.001继续训练15次，经过5分钟左右的训练，该模型对于训练集已经拥有80%以上的正确率，如图 5‑10图 5‑11所示

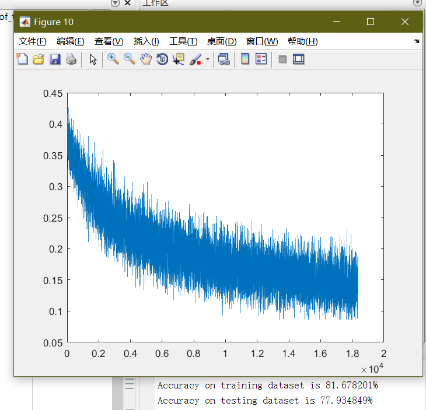


图 5‑10 主要基于ReLU的神经网络模型25次训练以后costfunction图线

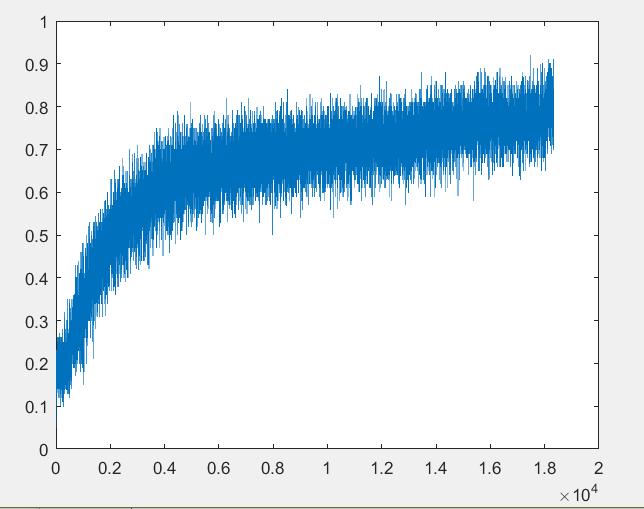


图 5‑11 主要基于ReLU的神经网络模型25次训练以后Accuracy图线

在此基础上，又训练了30min左右，继续加强该网络的提取特征的能力。观察何时可以达到上一个Sigmoid全连接神经网络的正确率。

### 训练结果

最终该模型实现了在15分钟内达到训练集正确率88.3533%，测试集正确率82.3448%的效果，如所示，整体上比上一个模型学习能力有大幅提升，学习效率用时几乎是Sigmoid全连接的5倍以上。但是对于本数据集样本的识别难度，想要进一步提升正确率变得异常艰难。难以突破90%大关。

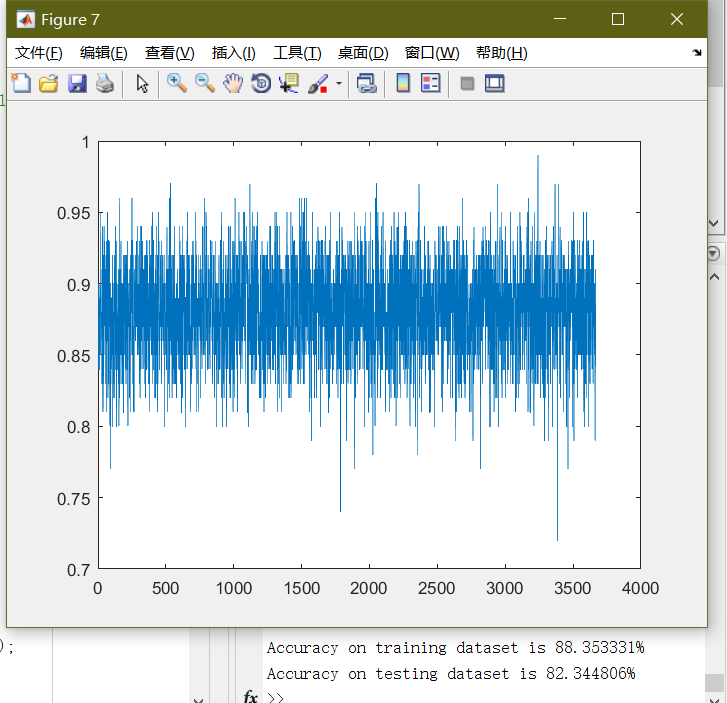


图 5‑12 最后一轮训练后的ACC图示和正确率

# 实验结果

经过本实验，实现了两种5层全连接神经网络的SVHN识别模型，训练集与测试集正确率均达到80%以上，其中基于4层ReLU配合最后1层Sigmoid的模型有较高的学习效率，可以在较少时间内实现梯度下降，达到同样，甚至更加的正确率的效果。

# 小结

基本实现了题目所需的功能，在更换激活函数、解决Bug、解决网络不学习问题的过程中，加深了对于神经网络的理解，整体对于深度学习原理有一个基本的认识。

神经网络的学习，难点在于对每一个参数的理解，以及cost function，梯度等等概念关系的辨识，才能解决当神经网络无法学习时，该从哪里入手哪里调参，同时在对比上，也深刻的理解到选择一个优秀的激活函数，对于神经网络训练的重要影响。

经过一个学期的学习，对于深度学习有了基本的了解与认识，掌握了全连接神经网络的具体运作方式，也对CNN等网络有了浅显的了解，希望在将来能了解更多前沿的深度学习理论与方法，并不断在实践中打磨自己，汲取更多的经验和知识，在人工智能浪潮席卷全球的特殊时期下，能抓住去往潮头的机会，争取能有一份对得起自己的学术成果。

# 代码

篇幅所限，只放最后一个效果最好的ReLU神经网络的训练代码，预处理，图片显示，前一个全连接Sigmoid神经网络，检查正确率等函数均在附件中一同提交。

## ReLU训练用trainmodle.m代码

clc;

clear;

load train\_1024.mat

load test\_1024.mat

L = 5;

layer\_size = [1024

512

256

64

10];

try

load model.mat

catch

for l = 1:L-1

% w{l} = 0.001 \* abs(randn(layer\_size(l+1,1), sum(layer\_size(l,:))));

w{l} = 0.1 \* randn(layer\_size(l+1,1), sum(layer\_size(l,:)));

end

fprintf('inital\n');

end

alpha = 0.00001;

J = []; % array to store cost of each mini batch

Acc = []; % array to store accuracy of each mini batch

max\_epoch = int32(input('Enter the max\_epoch:'));

mini\_batch = 100; % number of sample of each mini batch

figure % plot the cost

for iter=1:max\_epoch

fprintf('========No.%d training starts======= \n',iter);

idxs = randperm(train\_size);

for k = 1:ceil(train\_size/mini\_batch)

start\_idx = (k-1)\*mini\_batch+1;

end\_idx = min(k\*mini\_batch, train\_size);

if mod(k,10) == 0

percent = end\_idx/train\_size \*100;

fprintf('No.%d trains has complete %8.4f %% \n', iter,percent);

end

a{1} = X\_train(:,idxs(start\_idx:end\_idx));

y = trainlabels(:, idxs(start\_idx:end\_idx));

[a{2},z{2}] = fc(w{1},a{1});

[a{3},z{3}] = fc(w{2},a{2});

[a{4},z{4}] = fc(w{3},a{3});

[a{5},z{5}] = fc\_sig(w{4},a{4});

% [a{6},z{6}] = fc\_sig(w{5},a{5});

% [a{7},z{7}] = fc\_sig\_sig(w{6},a{6});

delta{L} = (a{L} - y).\* a{L} .\*(1-a{L});

% delta{6} = bc\_sig(w{6}, z{6}, delta{7});

% delta{5} = bc\_sig(w{5}, z{5}, delta{6});

delta{4} = bc(w{4}, z{4}, delta{5});

delta{3} = bc(w{3}, z{3}, delta{4});

delta{2} = bc(w{2}, z{2}, delta{3});

% delta{2} = bc\_sig(w{2},z{2},delta{3});

for l=1:L-1

grad\_w = delta{l+1} \* a{l}';

w{l} = w{l} - alpha\*grad\_w;

end

J = [J 1/length(y)\*sum(cost(a{L}, y))];

acc = accuracy(a{L}, y);

Acc =[Acc acc];

plot(J);

pause(0.000001);

end

fprintf('No.%d acc is %8.4f %% \n',iter, acc\*100);

end

figure

plot(Acc);

save model.mat w layer\_size;

% check acc of both

a{1} = X\_train;

[a{2},z{2}] = fc(w{1},a{1});

[a{3},z{3}] = fc(w{2},a{2});

[a{4},z{4}] = fc(w{3},a{3});

[a{5},z{5}] = fc\_sig(w{4},a{4});

train\_acc = accuracy(a{L}, trainlabels);

fprintf('Accuracy on training dataset is %f%%\n', train\_acc\*100);

% test data ACC

a{1} = X\_test;

[a{2},z{2}] = fc(w{1},a{1});

[a{3},z{3}] = fc(w{2},a{2});

[a{4},z{4}] = fc(w{3},a{3});

[a{5},z{5}] = fc\_sig(w{4},a{4});

test\_acc = accuracy(a{length(w)+1},test\_label);

fprintf('Accuracy on testing dataset is %f%%\n', test\_acc\*100);

## ReLU神经网络函数

### fc.m (ReLU的正向传播)

function [a\_next, z\_next] = fc(w, a)

f = @(s) (s>0).\*s;

z\_next = w \* a;

a\_next = f(z\_next);

end

### fc\_sig.m （Sigmoid的正向传播）

function [a\_next, z\_next] = fc(w, a)

f = @(s) 1 ./ (1 + exp(-s));

z\_next = w \* a;

a\_next = f(z\_next);

end

### bc.m (ReLU的反向传播)

function delta = bc(w, z, delta\_next)

f = @(s) max(0,s);

df = @(s) (s>0);

delta = (w' \* delta\_next) .\* df(z);

end

### bc\_sig.m （Sigmoid的反向传播）

function delta = bc(w, z, delta\_next)

f = @(s) 1 ./ (1 + exp(-s));

df = @(s) f(s) .\* (1 - f(s));

delta = (w' \* delta\_next) .\* df(z);

end

### cost.m （cost function）

function [J] = cost(a, y)

J = 1/2 \* sum((a - y).^2);

% J = -(y.\*log(a) + (1-y).\*log(1-a)) / length(y);

end

# 参考文献

[1] 斯坦福大学UFLDL深度学习教程. http://ufldl.stanford.edu/housenumbers/, 2011

[2] Yann LeCun.MNIST手写体数字数据集. http://yann.lecun.com/exdb/mnist/,2020

[3] 斯坦福大学UFLDL教程.http://ufldl.stanford.edu/housenumbers/, 2011

[4] LLLiuye. 博客园文章神经网络中常用的几种激活函数的理解. https://www.cnblogs.com/lliuye/p/9486500.html.2018