基于人工神经网络的在线识别Web应用

摘 要

近年来，随着机器学习的兴起，人工智能被应用到了生活中的方方面面。本文针对人工神经网络，主要设计并训练了一个用于识别26个大小写英文字母的模型，调整参数选出最优模型。接着，搭建服务器，整合模型和接口，完成后端配置。然后，开发一个微信小程序作为前端入口。最终，将应用部署上线。

**关键词：**人工神经网络 后端开发 前端开发

**Online Web Application Based on Artificial Neural Network**

**Abstract**

In recent years, with the rise of machine learning, artificial intelligence has been applied to all aspects of life. In this paper, an artificial neural network is designed and trained to identify a model of 26 English letters in upper and lower case. The parameters are adjusted to select the optimal model. Next, build the server, integrate the model and interface, and complete the back-end configuration. Then, develop a WeChat applet as a front-end portal. Finally, deploy the application deployment.

**Key Words:** Artificial neural networks; Backend development; Front-end development

目 录

摘要…………………………………………………………………………………Ⅰ

ABSTRACT…………………………………………………………………………Ⅱ

[第一章 绪论 1](#_Toc515515844)

[1.1 研究背景及研究意义 1](#_Toc515515845)

[1.2 人工神经网络历史及现状 1](#_Toc515515846)

[1.3 本文主要内容 1](#_Toc515515847)

[第二章 预备知识 2](#_Toc515515848)

[2.1 神经网络 2](#_Toc515515849)

[2.1.1 神经元 2](#_Toc515515850)

[2.1.2 神经网络模型 3](#_Toc515515851)

[2.1.3 求解神经网络 4](#_Toc515515852)

[2.2 npm包介绍 6](#_Toc515515853)

[2.2.1 Synaptic 6](#_Toc515515854)

[2.2.3 opencv4nodejs 6](#_Toc515515855)

[2.2.4 express 6](#_Toc515515856)

[第三章 神经网络的计算示例 7](#_Toc515515857)

[3.1 前向传播计算 7](#_Toc515515858)

[3.2 反向传播计算 8](#_Toc515515859)

[第四章 神经网络的构建 10](#_Toc515515860)

[4.1 准备图片库 10](#_Toc515515861)

[4.2 参数选择 11](#_Toc515515862)

[4.3 训练 12](#_Toc515515863)

[4.4 测试 14](#_Toc515515864)

[第五章后端部分 16](#_Toc515515865)

[5.1 接口配置 16](#_Toc515515866)

[5.1.1分析需求 16](#_Toc515515867)

[5.1.2获取二进制数据 16](#_Toc515515868)

[5.1.3数据转换 16](#_Toc515515869)

[5.2 模型配置 18](#_Toc515515870)

[5.3 https配置 18](#_Toc515515871)

[第六章 前端部分 18](#_Toc515515872)

[6.1 画板的实现 18](#_Toc515515873)

[6.2 接入后端接口 19](#_Toc515515874)

[6.2 应用结构 20](#_Toc515515875)

[第七章 结论 21](#_Toc515515876)

[参考文献 21](#_Toc515515877)

[附录 21](#_Toc515515878)

[digitsLoader.js 21](#_Toc515515879)

[labelsLoader.js 22](#_Toc515515880)

[rawMaker.js 23](#_Toc515515881)

[rawWriter.js 24](#_Toc515515882)

[emnist\_dl.js 24](#_Toc515515883)

[app-train.js 25](#_Toc515515884)

[emnist.js 26](#_Toc515515885)

[testnet.js 30](#_Toc515515886)

[app-web.js 31](#_Toc515515887)

[index.wxml 32](#_Toc515515888)

[index.js 33](#_Toc515515889)

# 第一章 绪论

## 1.1 研究背景及研究意义

近年来，由于机器学习的兴起，人工智能开始改变生活的方方面面。神经网络最先在语音识别领域获得了巨大成功，接着在图像和视觉领域也取得了广泛应用。大数据也是近年的热点之一，它背后的主要技术也是人工智能，如分析海量数据，助力决策等

人工智能正在引发新一轮的技术革命，影响到各行各业。现如今，深度学习成了最热门的技术，而神经网络则是深度学习的前身及基础。

当前，关于人工智能的开发几乎都基于python、C、matlab等语言，本文开创性的使用JavaScript作为开发语言，并成功训练出模型，其前后端的开发语言也都是JavaScript，实现了真正意义上的全栈开发，并为之后的应用开发流程提供了指导作用。

## 1.2 人工神经网络历史及现状

1943年，一个模拟神经元的简单的计算模型被提出，称为M-P模型，其所有的值都为0和1，可以模拟二进制逻辑。1956年，感知机概念被提出，在后来成为了人工神经网络的基础。

1969年，人们认为单层感知机不能解决异或问题，判定人工智能的研究没有价值，神经网络进入了第一次寒冬。1986年，BP算法使一层以上的网络进入实用阶段。1989年，神经网络被证明拟合能力接近无限，即任意决策边界都可以被逼近，神经网络迎来了第一次复兴。由于神经网络的一些缺陷，比如梯度计算不稳定，参数过多等问题，再加上SVM的兴起，在20世纪90年代中期，神经网络进入了第二次寒冬。在2006年，一种训练多层神经网络的方法被提出，神经网络又一次复兴，并更名为深度学习。

## 1.3 本文主要内容

本文主要设计一个多层感知机，并使用反向传播算法，完成神经网络模型的训练。再搭建服务器，接入模型，完成后端服务的配置。最后开发微信小程序，接入后端接口，部署应用。

本文研究内容为以下所示：

第一章为绪论部分。主要介绍本文研究背景现状；简要说明了神经网络的发展。末尾介绍了本论文所做工作及意义。

第二章为本文的预备知识。介绍了本文研究中所必备的一些前置条件和基础前置知识，主要介绍了神经网络的运作原理和反向传播的步骤，为后文证明提供依据。

第三章通过一个例子具体分析了神经网络运行过程中数据的变化过程。

第四章主要完成了神经网络的构建，包括图片库的准备，参数的选择及配置，模型的训练，最后测试模型正确率。

第五章主要完成后端部分的工作，包括图片的处理，接口配置，模型接入，https配置。

第六章主要完成了前端部分的工作，包括画板的实现，接入后端接口。

第七章则是对上述内容作了一个简要概括。

# 第二章 预备知识

## 2.1 神经网络

### 2.1.1 神经元

神经元是神经网络中最为基础的结构，它的输入是向量，线性组合输入的每个维度的值，再和一个阈值进行对比，若大于该值，则输出1，否则就输出-1。感知机与神经元有几点相似之处：第一，多输入，单输出；第二，以一个阈值分隔不同输出。

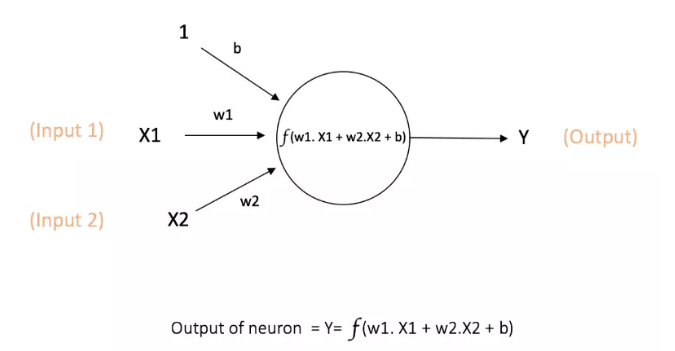


图2.1

神经元的结构里有两个最基本的成分：对输入向量的线性变换和对线性组合的结果进行阈值判断，实质是非线性变换。其实质上是仿射变换加一个非线性变换。而这个非线性变换，在机器学习领域中被叫做激活函数。

图2.1是一个以,和截距1作为输入的感知机，其输出为：

其中函数被称为激活函数。在本文中选用sigmoid函数作为激活函数：

其导数为

### 2.1.2 神经网络模型

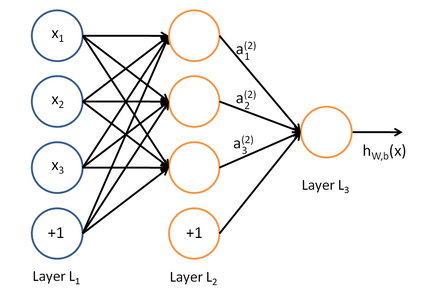


图2.2

许多单一神经元互相联结构成神经网络，一个神经元的输出是另一个神经元的输入，

如图2.2，最左边一层是输入层，最右边一层是输出层，中间左右的节点组成的一层是隐含层，标上“+1”的节点是偏置节点，也就是截距项。由于偏置节点不计入单元数，所以上例中有3个输入单元，3个隐含单元和1个输出单元。

一般用表示网络的层数，本例中，将第层记为，于是是输入层，是输出层。该神经网络有参数，其中是第层第单元与第层第单元之间的连接参数，即权重，是第层第单元的偏置项。偏置单元没有输入且总是输出。用表示第层的节点数，偏置单元不计入内。

用表示第层第单元的输出值，当时，，也就是第个网络输入值。对于给定参数集合，该网络可以按照公式（2,1）来计算输出结果，步骤如下：

以上计算步骤被称为前向传播。

除了图2.2的网络结构，神经网络还可以包含多个隐含层，即第1层是输入层，第层是输出层，中间的每个层与层紧密相连。这种模式下，也可以按照公式（2,1）计算，逐一计算第层的所有激活值，然后计算层，以此类推，直到第层。网络也可以有多个输出单元。由于这种联接没有闭环和回路，所以称为前馈神经网络。

### 2.1.3 求解神经网络

假设有一个样本集，包含个样例，则可以使用批量梯度下降法求解神经网络。对于单个样例，其代价函数为：

上式是一个方差代价函数。给定一个包含个样例的数据集，可以定义整体代价函数为：

公式（2,3）的第一项是均方差项，第二项是权重衰减项，作用是减小权重的幅度，防止过拟合。公式（2,2）是针对单个样例计算得到的方差代价函数，公式（2,3）是整体样本的代价函数，包含权重衰减项。

为了求解神经网络，需要针对参数和来求函数的最小值。先将每一个参数和初始化为一个接近零的随机值，之后对目标函数使用如梯度下降法的最优化算法。梯度下降法中每一次迭代都按照以下公式对参数进行更新：

式中是学习速率。其中的关键步骤是计算偏导数，反向传播算法是计算偏导数的一种有效算法。

由公式（2,3）（2,4）（2,5）可得：

由上式可知，只要先求出单个样例的代价函数的偏导数，就可推导出整体代价函数的偏导数。

反向传播算法的思路如下：给定一个样例，首先进行前向运算，计算出网络中所有的激活值，然后针对第层的每一个节点，可计算出残差，表明了改节点对最终值的残差产生了多少影响。对于最终的输出节点，可直接算出网络的激活值与实际值之差，将这个差距定义为。而对于隐含节点，将基于节点残差的加权平均值计算，具体步骤如下：

1. 利用前向传导公式（2,1），得到各层的激活值
2. 对于输出层，即第层的每个输出，可根据以下公式计算残差：
3. 对于的各层，第层的第个节点的残差可按照下式计算：

将公式（2,6）中与替换为与，可得：

以上逐次从前向后求导的过程即为反向传导。

1. 计算偏导数

至此，整体代价函数的偏导已可求得，将其带入公式（2,4）（2,5）即可求得每次迭代后更新的权重参数，进而迭代完成。

## 2.2 npm包介绍

npm是Node.js 的包管理器，它是js的开源库。

### 2.2.1 Synaptic

Synaptic是一个用于node.js和浏览器的神经网络库，它由原生js构建，无需任何依赖，可以构建和训练出任何一阶或二阶网络。

### 2.2.3 opencv4nodejs

Opencv4nodejs提供了全面的Nodejs绑定到OpenCV和OpenCV-contrib模块的API接口，可以使nodejs操作本地opencv，弥补了js在计算机视觉方面功能不足的缺点。

### 2.2.4 express

Express是最流行的基于nodejs的服务端框架，路由功能强大，性能优越，可扩展性强，几乎90%以上的node服务都基于express。

# 第三章 神经网络的计算示例

为进一步解释神经网络运作原理，下面给出一个例子。如图3.1所示，该网络具有3个输入，2个隐含和2两个输出。本例中使用单个训练集，给定输入分别为，期望输出分别为0.1,0.9，学习率此处给定为。一开始初始化时需要初始化和为随机值，这里给出的随机值分别为:

图3.1

## 3.1 前向传播计算

首先计算隐含层输入

然后使用sigmoid函数进行压缩

接下来计算实际的网络输出，和上方操作类似

下面计算总误差

## 3.2 反向传播计算

对：

先求

下面求

最后求

结合式（3.1）（3.2）（3.3）可得：

从当前权重中减去该值获得新的权重

重复以上步骤后，获得的新的权重如下：

传播到了隐含层

对

其中，因为h1影响到了两个输出

下面开始计算

得

同理

因此

继续求得

综上

则可更新为

同理

至此，所有权重已更新完毕，一轮迭代完成。

# 第四章 神经网络的构建

## 4.1 准备图片库

本文需要识别英文字母，所以需要26个英文字母的大小写手写图片数据。

关于神经网络，最著名的图片库是mnist，它由6万多张训练图片和1万多张测试图片组成，每张图片的长宽都是28像素，这些图片是由不同人手写的0到9的数字。

神经网络的训练离不开大量的数据，经过数十年的发展，已经积累了各式各样的数据集。我在以下网址中找到了英文字母的数据集：emnist。

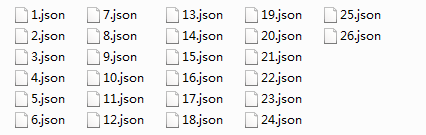
https://www.nist.gov/itl/iad/image-group/emnist-dataset

和mnist一样，emnist里图片的长宽也都是28像素。下载解压后里面是一系列压缩文件，这里需要emnist-letters-train-images-idx3-ubyte.gz和emnist-letters-train-labels-idx1-ubyte.gz，分别是二进制的图片数据和标记图片对应字母的映射。

二进制的数据无法直接拿来用，因此需要将原始数据处理成合适的格式，这里我做了一个转换器，可以将所有二进制数据转换成26个json文件，每个json对应一个字母。

node --max\_old\_space\_size=2048 emnist --count 100

由于处理的数据量比较大，可能会超过node默认的允许最大内存值，所以需要手动设置最大内存为一个比较大的值，这里设为了2G。--count表示为每个字母最多转化100个图片数据，所以转化后，最多有2600个图片数据。下图是生成的json列表。



这时数据虽然能被读取到，可还是不方便用。考虑到方便性和灵活性，我又做了一个工具，将每个图片处理成1X784的一维数组，只要输入需要的图片数量，就会返回一个由这些一维数组构成的二维数组和相应的结果。

至此图片库已经准备好了，只需要引入emnist.js，调用emnist.set(n1, n2)，就会返回n1条训练数据和n2条测试数据。

## 4.2 参数选择

首先，必须确定该网络需要多少个输入和输出神经元。由于每个图像的大小为28x28px，因此网络必须输入的像素数为28 x 28 = 784，这就是为什么我要将图片数据处理成1x784的数组的原因。而识别结果应为26个英文字母中的一个，因此输出神经元的数量为26。此外，网络应该至少有一个隐藏层，隐含层会减小误差同时也会增加计算量，考虑到本文的问题并不是特别复杂，所以我在这里设置了1层隐含层。

隐含层中的节点数也需要设置，隐含层节点数会直接影响到模型的性能，训练时会出现过拟合一般都是它的原因。对于隐含层节点数的选择，目前有一些公式可以大概估算出来，可是不同公式算出来的结果相差比较大，现在也没有一个精确的方法能计算出来，所以一般是先计算出一个大概的范围，然后根据实际情况，在满足计算精度的情况下，取尽可能少的点数。在本文中，结合一些经验公式和具体情况，隐含层节点取100.

另外，还有一些其他参数需要设置，在一开始，设一个大概的值，在之后悬链的时候，会根据训练的时长和模型的效果等情况进行调整。

学习率rate：0.1

最大迭代次数iterations：1000

最小错误率error：0.1

成本函数cost：Trainer.cost.CROSS\_ENTROPY

参数走到最优值的速度由学习率决定，如果训练时长时间不能收敛，可能是遇到了局部最优的情况，这时的学习率可能过小，可以适当调大。如果训练时函数无法收敛或是发散，那么可能是学习率过大，导致超过了最优值，此时应该调小学习率，使函数收敛。

设置一个合适的学习率，需要进行不断尝试。在一开始的时候，可以稍微设大一点，这样权重会改变的快一些，在迭代到一定的次数之后再减小学习率，看看变化。

随着迭代次数的增加，模型在训练集上的偏差会越来越小，但也有可能在达到最小偏差后不降反升。由于本次训练数据比较大，有近两千张图片，最大迭代次数设大一点比较好。

最小错误率表示当网络的误差小于该值时，则代表训练完成，就会停止训练，一般设为0.1。

## 4.3 训练

参数配置好后就可以开始训练了，由于训练需要几天时间，训练过程中CPU一直是100%，对电脑是个不小的损耗，为了避免过热死机等因素破坏训练过程，我把代码传到了云服务器上，在阿里云ECS上跑，由于是云服务器，稳定性较高，而且安全可靠，可以远程监控，当CPU占用量由100%降到5%以内时，就代表训练完成了。

我是用PuTTY远程连接服务器的，连接后只能以命令行的形式操作。先用FTP工具将资源文件和代码上传到服务器上，再用node运行程序。

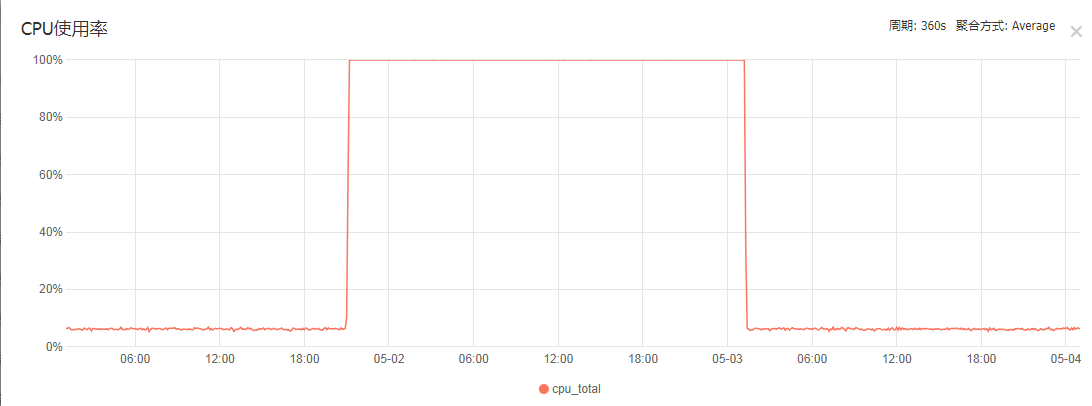
可是由于是远程连接，一旦断开连接，当前跑的程序就会中断，因此为了让程度能在服务器上不间断的跑，需要用到守护进程，这里我选择forever。forever的作用是让node服务能在服务器上长时间的运行，forever有一个自己的进程，每守护一个程序，forever就会在自己的进程下开一个子进程，只要主进程不挂，子进程就不会有事。

Forever还有一些配置项，这里我用了以下配置：

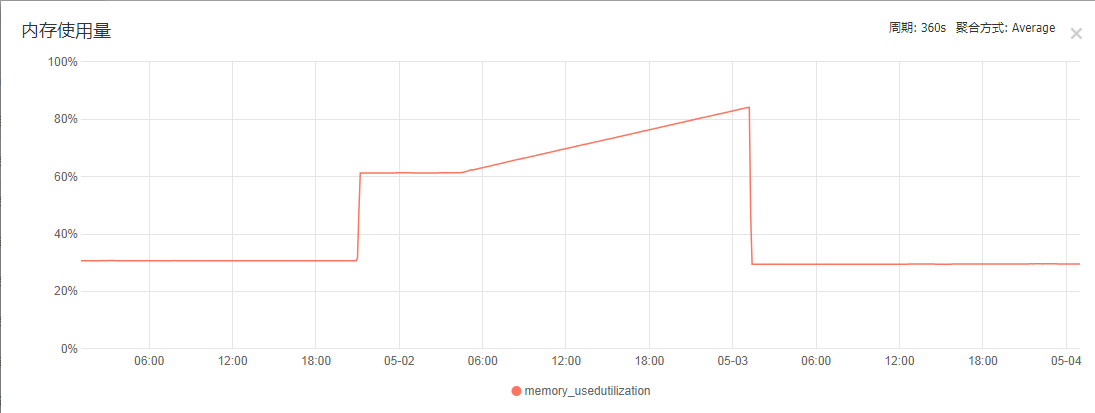
forever start -o train.log -m 1 app-train.js

意思是开始执行app-train.js文件，将输出打印到日志文件train.log中，最大执行次数为1次。通过安装主机监控插件，云监控可以采集服务器监控数据，展示在控制台上。在训练过程中，可以随时监控服务器资源使用情况。

下图是一次完整训练过程的监控图表，该云服务器的配置是1核2GB内存，系统是CentOS 7.3。可以看到CPU是一直是满负荷运行，说明训练时消耗的计算量是非常大的。内存占用量一开始不变，之后开始缓慢上升，这是明显的内存泄漏，由于某些原因js的垃圾回收机制并未回收某些资源导致内存占用越来越大。







从图中可以看出是在训练一段时间后内存才开始上升，所以应该是所用库synaptic的缺陷。这个问题有两种应对方法，一是增大内存，只要内存足够大，泄漏的这点内存就可以忽略不计，但这显然是治标不治本；第二种方法可以从源头解决问题，就是修改源码，找出泄漏的源头，一般是引用类型使用完后没有赋值为null。我已将问题反映给作者，期待他的回复。

训练过程中，每经过一次迭代，都会打印一条日志，如：

iterations 156 error 0.863792625747795 rate 0.07

iterations 157 error 0.7917011813679228 rate 0.07

iterations 158 error 0.7152523206229313 rate 0.07

iterations 159 error 0.6988286124934737 rate 0.07

iterations 160 error 0.6068983196793305 rate 0.07

iterations 161 error 0.5854829950364042 rate 0.07

……

意思是当前迭代次数，当前错误率，学习率。通过日志，可以看出训练的情况，从而调整参数优化网络。

经过调整了多次参数并分别进行了训练，我总结了一下几条规律：

1. 整体来看，错误率是随着迭代次数增加而减小的，而且错误率越小，下降的越慢。
2. 一次完整的训练一般会有上百次迭代，初始错误率一般在十几到几十之间，初始错误率越大，迭代次数越多，需要的时间也越长。
3. 样本数量与单次迭代所需的时间呈正相关；初始错误率与学习率呈正相关。
4. 训练时间 = 迭代次数 × 单次迭代所需的时间，一次完整的训练一般需要一到两天。



## 4.4 测试

训练完成后生成的模型是一个js文件，6M左右，里面是大量的网络参数。由于不知道训练是否成功，所以训练完成后要做的第一件事就是测试模型的质量。这里要用到上文提到的emnist工具，传入需要测试的图片数量，返回相应的数据，将这些数据传入模型，获取识别结果，再将该结果与原数据的正确结果比对，从而得出识别率。识别率越大，则模型训练的就越好。

let data = fs.readFileSync('./nets/net0-07.js', 'utf8')

const net = eval('(' + data + ')');

function getMaxIndex(arr) {

let max = 0,

ret = -1;

arr.forEach(function(item, index) {

if (item > max) {

max = item;

ret = index

}

})

return {max: max, i: ret}

}

let num = 0, // 总数

suc = 0， // 成功数

now = new Date();

testSet.forEach(function(item, index) {

let ret = net(item.input);

let res1 = getMaxIndex(ret)

res2 = getMaxIndex(item.output)

num++;

if (res1.i === res2.i) {

suc++;

}

})

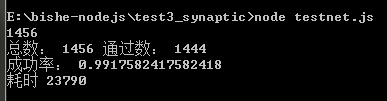
console.log('总数：', num, '通过数：', suc)

console.log('成功率：', suc/num)

console.log('耗时', new Date() - now)

取出测试数据后就可以通过以上代码测试，将输出测试总个数，正确识别数，识别成功率，以及总耗时。、

为了统一，我将测试个数设为1456个。



上图测试的是学习率为0.07，训练个数为2080训练出来的模型，从图中可以看出两点信息：

1. 该模型准确率是相当的高。
2. 单条数据的平均识别时间为16.3毫秒，几乎可以说是实时出结果，速度是相当的快。

评价一个模型的好坏无非从准确率和速度两点出发，因此该模型可以说是非常成功。

下表为其他配置相同，仅学习率不同的情况下训练出来的模型的准确率：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学习率 | 测试总数 | 正确总数 | 测试耗时(ms) | 正确率(%) |
| 0.01 | 1456 | 1411 | 25335 | 96.90934 |
| 0.03 | 1456 | 1449 | 25044 | 99.51923 |
| 0.07 | 1456 | 1444 | 23790 | 99.17582 |

# 第五章后端部分

## 5.1 接口配置

### 5.1.1分析需求

在写接口之前，需要明确一下需求：前端返回图像的二进制数据，后端返回json格式的识别结果，接口需要遵循RESTful规范。

根据以上条件，可以分析出以下结论。

1. 数据的交互中有二进制数据和json数据，因此不能使用get方式，最好使用post请求。
2. 由于RESTful规范，需要将接口路由统一配置，接口名尽可能的语义化，因此将接口定为/api/imgdata

### 5.1.2获取二进制数据

由于前端传进来的是长宽28像素的二进制图像数据，不同于一般的请求，所以需要使用multiparty这个npm包来获取图像数据。在nodejs中，要使用一个包，必须在一开始用require引入进来，因此需要把multiparty引入后并保存为变量以供之后使用。

const multiparty = require('multiparty')

在接口中，需要实例化一个multiparty对象

const form = new multiparty.Form()

取到请求数据后使用form.parse(res,cb(err,fileds,files))，回调函数的第三个参数files中就存放着二进制数据。通过files.data[0].path能成功取到二进制数据。

### 5.1.3数据转换

由于模型的输入是1x784的一维数组，所以还需要一个二进制转数组的步骤。这里用到了另一个关键的npm包opencv4nodejs，这个包实际上是opencv的js版，安装它之前需要先安装opencv和python，它提供了一系列api，通过调用它的api来调用本地的opencv进行图像处理操作。

同样，使用前需要先引入：

const cv = require('opencv4nodejs')

通过cv.imreadAsync(filePath,flags,callback(err,img))读取数据，并在回调函数中进行进一步处理。其中，filePath是图片数据，flags是读取方式，为了方便，这里取cv.IMREAD\_GRAYSCALE，表示读取后先进行灰度化处理。

普通彩图的RGB三位的值不同，所以会显示出不同的颜色，而灰度化后像素的RGB三个值是相等的，图像就没有了颜色，整体上显灰色，只有亮度上的不同，所以叫做灰度化。灰度值的范围是0到255。值越大，图像越白；灰度值越小，图像越黑。

在回调函数中，图像数据是cv中特有的Mat格式，之后所有对它的操作都基于Mat。后端接口不仅要实现需求，还要尽可能的考虑其他异常情况。为了防止前端图像尺寸传错，需要在一开始加一层保险，即对图像重新设置大小为28像素。使用getDataAsArray函数将Mat转化为28x28的矩阵，再将矩阵进行操作拉伸成1x784的矩阵。

到现在为止，仅仅是把数据转化成了符合要求的格式，可是里面的值要进行怎样的处理还不清楚。这里使用反推法，对训练数据执行以上相反的操作，转化为图片显示出来。

下图是原始的g，可以看出原始的数据并不是正的，而是先将字母g进行y轴镜像翻转，再顺时针旋转了90度。

D:\Documents\大四下\毕设文档\论文\imgs\g.png

因此在接口中也要执行相同的操作：

img = img.flip(1);

let rotate = cv.getRotationMatrix2D(new cv.Point(13.5,13.5), 90);

img= img.warpAffine(rotate);

此外，训练数据中的值都在0-1之间，所以还需要进行二值化，可对每个值进行以下操作：

Number(Math.abs((Number(arr[i]-255))/255).toFixed(3))

整个转化流程如下：

最后得到的一维数组就可以传入模型，模型识别非常快，几毫秒就能返回结果，结果是一个1x26的数组，代表26个英文字母，每项的值代表对应字母的可能性，越接近1，则越可能是该字母。一般来说，如果识别成功的话，26个值里只有1项无限趋近于1，而其他25个值无限趋近于0.

## 5.2 模型配置

由于模型是一个可以独立运行的js文件，但在一个js文件里并不能直接执行另一个js文件，光是读取并没有什么用。所以这里需要使用eval语句，先用nodejs的模块以utf8的格式读取模型文件为字符串，再使用eval将字符串解析为语句并执行，具体写法如下：

let netData = fs.readFileSync('./nets/net0-07.js', 'utf8')

const net = eval('(' + netData + ')');

其中，net就是读取到的模型，只要将其放入接口中，传入数据，就会返回1x26的识别结果。

## 5.3 https配置

配置HTTPS需要https证书，而在申请证书之前，需要先申请域名，并做好备案，域名指向和解析到服务器。

阿里云提供免费HTPPS证书，所谓的https证书其实就是几个文件。使用Express框架配置https只需要两个文件：private.key: 私钥和csr.pem: CSR证书签名。

先以utf8格式读取这两个文件，引入https模块，在以https模块启动服务的时候，传入以上两个文件的配置。

由于https的端口是443，所以还需要在服务器上开放443端口，另外本应用的后端只开放443端口。

# 第六章 前端部分

## 6.1 画板的实现

前端部分需要实现的主要功能是用户手写字母，并将图像发送给后端。这里使用了html中的canvas标签，它有很多功能，主要是它是用来绘制客户端的矢量图形。它只是一块画布，但是有很多api给js使用，所以可以通过脚本绘制图像到画布上。

canvas标签最初在 Safari浏览器中由Apple引入，它仅仅是一个图形的容器，具体的绘制需要js来实现。这里我需要实现一个画板的功能，首先获取canvas上下文，之后的所有绘图操作都要通过它来实现，接下来一共有三步：

1. 点击、按下事件、

当鼠标点击或手指按下时，保存当前点的坐标，并初始化画图的配置，画笔颜色、粗细，平滑度等。

1. 移动事件

获取移动时点的坐标，与之前保存的坐标连接，笔触移动到新的坐标上，并且用新坐标覆盖原来保存的坐标。每触发一次移动事件就执行一次以上操作。

1. 松开事件

中断轨迹，结束绘制。

除了绘制操作，还需要清空画布的功能。清空画布有几种实现方法，这里使用覆盖法。即画一个与画布等高宽的白色背景的矩形，这样之前画的所有轨迹就都被覆盖掉了。至此，一个简单的画板就实现了。

## 6.2 接入后端接口

在接接口之前，需要获取画布内容。小程序提供的wx.canvasToTempFilePath方法可以将指定大小的画布区域中的内容生成图片，并返回图片路径。本文中需要将整个画布生成28像素宽高的图片。

获取到图片路径后需要调接口上传图片数据，小程序提供了uploadFile方法，以post方式发送请求，接口地址为<https://sadxu.top/api/imgdata>，

wx.uploadFile({

url: 'https://sadxu.top/api/imgdata',

filePath: url,

name: 'data',

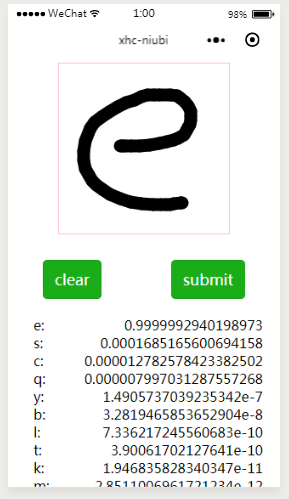
success: function (res) {},

fail: function (err) {}

})

接口的返回分为成功和失败两种情况，失败时在fail中处理，一般给出提示；成功时在success中处理，取到返回体中的数据，是1x26的一维数组，再将这些数据展示到界面上，即每个字母对应的识别率是多少。

小程序截图如下：



关于前端，我既做了网页的形式，也做了小程序的形式。两者各有优缺点，网页在手机上和电脑上都可以使用，写起来比较自由，就是比较复杂，而且在手机上体验不是特别好。而小程序只能在微信里使用，但做起来比较简单，用户体验接近原生app，缺点自由度不高，只能调用官方提供的接口，无法引入外部包，而且需要审核通过后才能使用。另外为了网络的安全性，微信强制规定所有小程序的后端接口都需要https加密。

## 6.2 应用结构

至此一个完整的应用就完成了，该应用结构如下图所示：

传入图像数据

发起请求

模型

后端

前端

返回识别结果

返回响应

# 第七章 结论

本论文是关于神经网络识别应用的研究与开发。内容主要为三个部分：第一部分通过分析具体情况完成神经网络的设计和训练，根据训练情况调整参数，最后获得最优模型。第二部分是将模型嵌入后端，根据实际情况，设计接口。第三部分是完成微信小程序的开发，接入后端接口。一个应用，这三大部分环环相扣，缺一不可。

全栈开发是当今广受讨论的理念，该应用的开发全部基于JavaScript，不光是前后端，还包括算法部分，实现了真正意义上的全栈。本文展现了js近乎全能的优势，也在一定程度上推动了应用开发方式的变革。

人工智能是现在最火热的领域之一，迄今为止已经获得了许多成果并在各行各业都有应用，当然它仍有一些不足，相信在未来，会实现真正的人工智能。

# 参考文献

[1]justadudewhohacks.Opencv4nodejs[EB/OL].<https://github.com/justadudewhohacks/opencv4nodejs> .2017.3

[2] Cazala.synapptic.Synaptic [EB/OL]. <https://github.com/cazala/synaptic>.2017.10.

[3]UjjwalKarn.AQuickIntroductiontoNeuralNetworks[EB/OL].https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks 2016.8.

[4]MarkFeng.浏览器使用synaptic.js训练简单的神经网络推荐系统[EB/OL]. <https://github.com/markselby9/ml-in-browser>.2017.10.

[5] 微信公众平台. 小程序开发文档[EB/OL]. <https://developers.weixin.qq.com/miniprogram/dev/index.html?t=2018518> .2018.5.

# 附录

## digitsLoader.js

const Q = require('q')

module.exports = function (labelFileName = './data/emnist-letters-train-images-idx3-ubyte') {

const fs = require('fs'),

deferred = Q.defer();

digits = [];

// fs,readStream

const stream = new fs.ReadStream(labelFileName);

let ver = 0, digitCount = 0, x = 0, y = 0, start = 0;

stream.on('readable', function () {

let buf = stream.read();

if (buf) {

if (ver != 2051) {

ver = buf.readInt32BE(0);

console.log(`DB digits Version: ${ver}`);

digitCount = buf.readInt32BE(4);

console.log(`Total digits: ${digitCount}`);

x = buf.readInt32BE(8);

y = buf.readInt32BE(12);

console.log(`x x y: ${x} x ${y}`);

start = 16;

}

for (let i = start; i< buf.length; i++) {

digits.push(buf.readUInt8(i));

}

start = 0;

}

});

stream.on('end', function () {

deferred.resolve(digits);

});

return deferred.promise;

}

## labelsLoader.js

const Q = require('q')

module.exports = function (labelFileName = './data/emnist-letters-train-labels-idx1-ubyte') {

const fs = require('fs'),

deferred = Q.defer();

labels = [];

// fs,readStream

const stream = new fs.ReadStream(labelFileName);

let ver = 0, labelCount = 0, start = 0;

stream.on('readable', function () {

let buf = stream.read();

if (buf) {

if (ver != 2049) {

ver = buf.readInt32BE(0);

//console.log(`DB Labels Version: ${ver}`);

labelCount = buf.readInt32BE(4);

//console.log(`Total labels: ${labelCount}`);

start = 8;

}

for (let i = start; i< buf.length; i++) {

labels.push(buf.readUInt8(i));

}

start = 0;

}

});

stream.on('end', function () {

//console.log(labels);

deferred.resolve(labels);

//console.log('finish');

});

return deferred.promise;

}

## rawMaker.js

module.exports = function (labels,digits,count) {

let raw = [];

const imageSize = 28\*28,

normalize = function (num) {

if (num != 0) {

return Math.round(1000/(255/num))/1000;

} else {

return 0;

}

};

count = count || labels.length;

for (let i in labels) {

if (i >= count) {

break;

}

let start = i\*imageSize;

if (! Array.isArray(raw[labels[i]])) {

raw[labels[i]] = [];

}

let range = digits.slice(start,start+imageSize).map(normalize);

raw[labels[i]].push(...range);

if (i % 1000 == 0) {

console.log(`Pass ${i} items...`);

}

}

console.log(`Finish processing ${count} items...`);

return raw;

}

## rawWriter.js

const Q = require('q')

module.exports = function (raw, digitsDir = './digits') {

const fs = require('fs');

for (let i in raw) {

console.log(`Start make "${i}.json with ${raw[i].length/(28\*28)} images"`);

let wstream = fs.createWriteStream(`${digitsDir}/${i}.json`);

wstream.write('{ "data": [' + raw[i].join(',') +']}');

wstream.end();

}

};

## emnist\_dl.js

"use strict"

/\* 生成数据 \*/

// node --max\_old\_space\_size=2048 emnist --count 100

const co = require('co'),

argv = require('optimist').argv,

labelsLoader = require('./lib/labelsLoader'),

digitsLoader = require('./lib/digitsLoader'),

rawMaker = require('./lib/rawMaker'),

rawWriter = require('./lib/rawWriter');

co(function\* () {

let labels = yield labelsLoader(),

count = argv.count || labels.length,

digits = yield digitsLoader();

console.log('step1')

console.log(labels.length);

console.log(digits.length);

// console.log(digits.length/(28\*28));

let raw = rawMaker(labels,digits,count);

rawWriter(raw);

}).catch(function (err) {

console.error(err.stack);

});

## app-train.js

const fs = require('fs')

/\* 准备数据 \*/

const emnist = require('./emnist')

const set = emnist.set(2080, 26) // 2080 训练 78 测试

const trainingSet = set.training

const testSet = set.test

/\* 构建网络 \*/

const synaptic = require('synaptic')

const Layer = synaptic.Layer

const Network = synaptic.Network

const Trainer = synaptic.Trainer

const inputLayer = new Layer(784)

const hiddenLayer = new Layer(150)

const outputLayer = new Layer(26)

inputLayer.project(hiddenLayer)

hiddenLayer.project(outputLayer)

const myNetwork = new Network({

input: inputLayer,

hidden: [hiddenLayer],

output: outputLayer

})

console.log('start train...')

const trainer = new Trainer(myNetwork)

trainer.train(trainingSet, {

rate: 0.09,

interations: 1000, // 迭代次数

error: 0.1, // 最小错误

shuffle: true, // 随机排序

log: 1, //

cost: Trainer.cost.CROSS\_ENTROPY

})

console.log('finish train...')

// 保存训练好的网络

let standalone = myNetwork.standalone();

fs.writeFile("./net0-09.js", standalone, function(err) {

if (err) {

console.log(err)

} else {

// logger.info("The file was saved!")

console.log("The file was saved!");

}

});

## emnist.js

// MNIST digits

var MNIST = [];

// 图片尺寸 28 x 28

var size = 28;

// 原始数据

var raw = [

require('./data/1.json').data,

require('./data/2.json').data,

require('./data/3.json').data,

require('./data/4.json').data,

require('./data/5.json').data,

require('./data/6.json').data,

require('./data/7.json').data,

require('./data/8.json').data,

require('./data/9.json').data,

require('./data/10.json').data,

require('./data/11.json').data,

require('./data/12.json').data,

require('./data/13.json').data,

require('./data/14.json').data,

require('./data/15.json').data,

require('./data/16.json').data,

require('./data/17.json').data,

require('./data/18.json').data,

require('./data/19.json').data,

require('./data/20.json').data,

require('./data/21.json').data,

require('./data/22.json').data,

require('./data/23.json').data,

require('./data/24.json').data,

require('./data/25.json').data,

require('./data/26.json').data

];

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 ,22, 23, 24, 25, 26].forEach(function (id) {

// mnist digit

var digit = {

id: id - 1

};

// raw data

digit.raw = raw[digit.id];

// 数量取28的倍数

digit.length = digit.raw.length / (size \* size) | 0;

digit.get = function (\_which) {

var which = \_which;

if ('undefined' == typeof which || which > digit.length || which < 0) {

which = Math.random() \* digit.length | 0;

}

var sample = [];

for (

var length = size \* size,

start = which \* length,

i = 0;

i < length;

sample.push(digit.raw[start + i++])

);

return sample;

}

digit.range = function (start, end) {

if (start < 0)

start = 0;

if (end >= digit.length)

end = digit.length - 1;

if (start > end) {

var tmp = start;

start = end;

end = tmp;

}

var range = [];

for (

var i = start;

i <= end;

range.push(digit.get(i++))

);

return range;

}

digit.set = function (start, end) {

var set = [];

var output = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0 ];

output[digit.id] = 1;

var range = digit.range(start, end);

for (

var i = 0;

i < range.length;

set.push({

input: range[i++],

output: output

})

);

return set;

}

MNIST.push(digit);

});

MNIST.set = function (\_training, \_test) {

var training = \_training / 26 | 0;

var test = \_test / 26 | 0;

if (training < 1)

training = 1;

if (test < 1)

test = 1;

var trainingSet = [];

var testSet = [];

for (var i = 0; i < 26; i++) {

trainingSet = trainingSet.concat(MNIST[i].set(0, training - 1));

testSet = testSet.concat(MNIST[i].set(training, training + test - 1));

}

return {

training: shuffle(trainingSet),

test: shuffle(testSet)

}

}

// 打乱数据

function shuffle(v) {

for (var j, x, i = v.length; i; j = parseInt(Math.random() \* i), x = v[--i], v[i] = v[j], v[j] = x);

return v;

};

/\*\* 导出 \*\*/

// CommonJS & AMD

if (typeof define !== 'undefined' && define.amd) {

define([], function () { return MNIST });

}

// Node.js

if (typeof module !== 'undefined' && module.exports) {

module.exports = MNIST;

}

// Browser

if (typeof window == 'object') {

(function () {

var old = window['mnist'];

MNIST.ninja = function () {

window['mnist'] = old;

return MNIST;

};

})();

window['mnist'] = MNIST;

}

## testnet.js

const fs = require('fs')

const cv = require('opencv4nodejs')

const emnist = require('./emnist')

const set = emnist.set(2, 1456)

const trainingSet = set.training

const testSet = set.test

console.log(testSet.length)

let data = fs.readFileSync('./nets/net.js', 'utf8')

const net = eval('(' + data + ')');

function getMaxIndex(arr) {

let max = 0,

ret = -1;

arr.forEach(function(item, index) {

if (item > max) {

max = item;

ret = index

}

})

return {max: max, i: ret}

}

let num = 0, // 总数

suc = 0; // 成功数

let now = new Date();

testSet.forEach(function(item, index) {

let ret = net(item.input);

let res1 = getMaxIndex(ret)

res2 = getMaxIndex(item.output)

num++;

if (res1.i === res2.i) {

suc++;

}

})

console.log('总数：', num, '通过数：', suc)

console.log('成功率：', suc/num)

console.log('耗时', new Date() - now)

## app-web.js

const https = require('https')

const fs = require('fs')

const express = require('express')

const bodyParse = require('body-parser')

const multiparty = require('multiparty')

const cv = require('opencv4nodejs')

const router = express.Router()

const app = express()

const httpsOptions = {

key: fs.readFileSync('./https/214400255360454.key', 'utf8'),

cert: fs.readFileSync('./https/214400255360454.pem', 'utf8')

};

app.use(bodyParse.json());

app.use(bodyParse.urlencoded({extended:true}))

let netData = fs.readFileSync('./nets/net0-07.js', 'utf8')

const net = eval('(' + netData + ')');

router.post('/imgdata', (req, res) => {

const form = new multiparty.Form()

form.parse(req, function(err, fields, files) {

if (err) {console.log(err)}

cv.imreadAsync(files.data[0].path,cv.IMREAD\_GRAYSCALE,(err, img) => {

img = img.flip(1); // 镜像旋转 0 x 1 y -1 x+y

let rotate = cv.getRotationMatrix2D(new cv.Point(13.5,13.5), 90) // 逆90

img= img.warpAffine(rotate)

let arr = img.getDataAsArray(); // mat -> arr

// 二维 -> 一维

arr = arr.join(',').split(',')

for (let i = 0; i < arr.length; i++) {

arr[i] = Number(Math.abs((Number(arr[i]-255))/255).toFixed(3));

}

// 识别

let ret = net(arr)

res.json({'err':0,'msg':ret})

})

})

})

app.use('/api', router);

https.createServer(httpsOptions, app).listen(443, function() {

console.log('listening port 443')

})

## index.wxml

<view class="container">

<canvas class="canvas" style="width: 200px; height: 200px;"

disable-scroll="true" canvas-id="drawMap"

binderror="canvasErrorCb" bindtouchstart="drawStart" bindtouchmove='drawMove' bindtouchend="drawEnd" bindtouchcancel="drawCancle"></canvas>

<!-- 按钮 -->

<view class="menu-wrapper">

<button type="primary" size='default' bindtap="clearCtx">clear</button>

<button type="primary" size="default" bindtap="getAndUpload">submit</button>

</view>

<!-- 结果 -->

<view class="result-wrapper" wx:if="{{resArr.length == 26}}">

<view class="result-item" wx:for="{{resArr}}">

<text>{{item[0]}}:</text>

<text>{{item[1]}}</text>

</view>

</view>

</view>

## index.js

Page({

data: {

ctx: null,

// drawFlag: false,

xy: [],

//

wordList: ['a','b','c','d','e','f','g','h','i','j','k','l','m','n',

'o','p','q','r','s','t','u','v','w','x','y','z'],

resArr: []

},

canvasErrorCb: function (e) {

console.log(e.detail.errMag)

},

onLoad: function () {

let ctx = wx.createCanvasContext("drawMap");

ctx.setLineWidth(15)

ctx.setLineCap('round')

ctx.setFillStyle('white')

ctx.fillRect(0, 0, 200, 200)

ctx.draw()

this.setData({

ctx: ctx

})

},

drawStart: function (e) {

this.data.xy = [e.touches[0].x, e.touches[0].y]

},

drawMove: function (e) {

let x1, y1;

x1 = e.touches[0].x;

y1 = e.touches[0].y;

let ctx = this.data.ctx,

xy = this.data.xy;

ctx.moveTo(xy[0], xy[1])

ctx.lineTo(x1, y1)

ctx.stroke()

ctx.draw(true)

this.data.xy = [x1, y1]

},

drawEnd: function (e) {

},

drawCancle: function (e) {

console.log('err')

console.log(e)

},

// 清空画布

clearCtx: function() {

let ctx = this.data.ctx;

this.setData({

resArr: []

})

ctx.setFillStyle('white')

ctx.fillRect(0, 0, 200, 200)

ctx.draw();

ctx.setLineWidth(15)

ctx.setLineCap('round')

},

// 获取并上传

getAndUpload: function () {

let imgUrl = '',

that = this;

wx.canvasToTempFilePath({

destWidth: 28,

destHeight: 28,

canvasId: 'drawMap',

success: function (res) {

console.log(res.tempFilePath)

that.postImg(res.tempFilePath)

},

fail: function (err) {

console.log(err)

}

})

},

postImg: function (url) {

const that = this;

wx.uploadFile({

url: 'https://sadxu.top/api/imgdata',

filePath: url,

name: 'data',

success: function (res) {

console.log(res.data)

let data = JSON.parse(res.data);

if (data.err == 0) {

that.showRes(data.msg)

} else {

console.log('err in success')

}

},

fail: function (err) {

console.log(err)

}

})

},

// 结果处理

showRes(data) {

const that = this;

if (data.length === 26) {

let arr = [];

data.forEach(function(item, index) {

arr.push([that.data.wordList[index], item])

})

arr.sort(function(a1,a2) {

return a2[1] - a1[1]

})

this.setData({

resArr: arr

})

console.log(arr)

}

}

})