基于人工神经网络的在线识别Web应用

目 录

摘要…………………………………………………………………………………I

ABSTRACT………………………………………………………………………II

第一章 引言………………………………………………………………………1

1.1应用结构及思路……………………………………………………………2

1.1.1为什么要用javascript……………………………………………2

1.1.2人工神经网络基础知识……………………………………………3

1.1.3 SVM的识别……………………………………………………………6

第二章 开发流程……………………………………………………………9

2.1 模型部分………………………………………………………………9

2.1.1 1. 准备图片库…………………………………………………12

2.1.2 构建网络………………………………………………………12

2.1.3 测试模型………………………………………………………12

2.2 后端部分………………………………………………………………9

2.2.1 搭建服务…………………………………………………10

2.2.2 对接模型…………………………………………………12

2.2.3 写接口……………………………………………………12

2.2.4 HTTPS配置…………………………………………………12

2.3前端部分………………………………………………………………14

2.3.1 交互页面…………………………………………………17

2.3.2 微信小程序…………………………………………………17

结语………………………………………………………………………………104

参考文献…………………………………………………………………………106

致谢………………………………………………………………………………107

该应用可分为三大部分：模型部分，前端部分和后端部分。

模型部分主要是准备图片库，构建网络，训练、测试，最终生成识别模型。

前端部分主要是完成前端页面的制作及接口对接数据展示，这里准备采用微信小程序的形式。

后端部分主要是搭建服务，对接模型，写接口，连通前端和模型。

整个应用的流程是这样的：用户打开微信小程序，画出某个字符，点击识别按钮，会将字符的数据发送给服务器，服务器接收到数据，进行一系列处理转化成模型可接收的数据形式并传给模型，模型输出识别结果，服务器将获取到的结果返回给前端，最后前端将结果展现出来，完成识别。

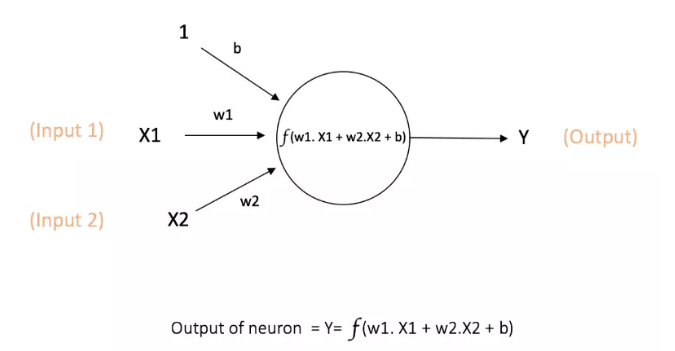
本应用主要用到的语言为javascript

为什么要用javascript

对于javascript，大多数人总是对它有所偏见，认为这是它只是个依附于浏览器的没多大用处的语言。我作为一个js的忠实粉丝，将一一列出js的过人之处。速度快。在V8引擎下，js已经快过了python和ruby，虽然和java、C++相比还是慢了不少，不过作为一个解释型语言，那也是没办法的事。Js也可以做矩阵操作。随着js的不断发展，各种库陆陆续续出现，比如矩阵操作方面，就有math.js。也可以做服务端开发，如nodejs。

基础知识

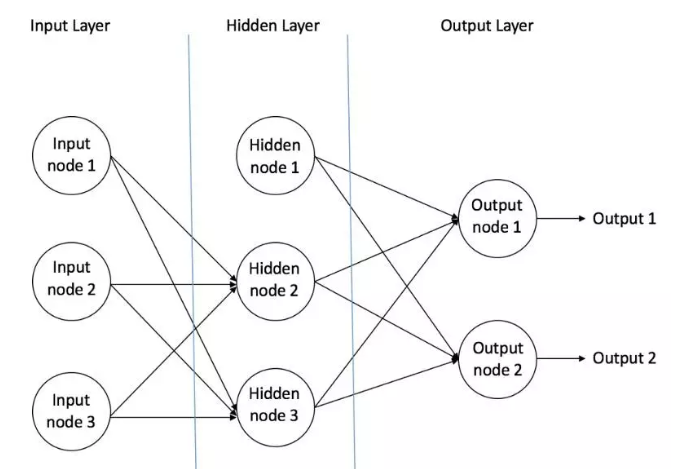
人工神经网络（ANN）是一种由人脑启发的计算模型。它由神经元组成，神经元是神经网络的基本单元。神经元从其它来源接收输入，每个输入分配一个权重，权重根据输入的重要程度赋予。神经元使用激活函数作用于所有输入的加权之和，然后给出输出。



上述网络包含输入X1和X2，与这些输入相关联的权重 w1和w2。还有另一个输入1 与权重b（即偏差）。神经元的输出Y如图所示。函数f是非线性的，称为激活函数。激活函数的作用是引入非线性因素，来解决线性模型所不能解决的问题。因为大多数现实世界的数据都是非线性的，而线性模型只能解决线性问题，所以引入激活函数是非常有必要的。

前馈神经网络

前馈神经网络是第一个也是最简单的人工神经网络。它包含多个分层排列的神经元（节点）。相邻层的节点之间有连接，所有这些连接都有与之相关的权重。



一个前馈神经网络由三种类型的节点组成：

输入节点：输入节点将来自外部的信息提供给网络，统称为“输入层”。在任何输入节点中都不含有计算，它们只是将信息传递给隐藏节点。

隐含节点：隐含节点与外界没有直接联系。它们主要执行计算并将信息从输入节点传输到输出节点。隐含节点的集合形成“隐含层”。虽然前馈网络只有一个输入层和一个输出层，但它可以具有零个或多个隐藏层。

输出节点：输出节点统称为“输出层”，负责计算并将信息输出到外部。

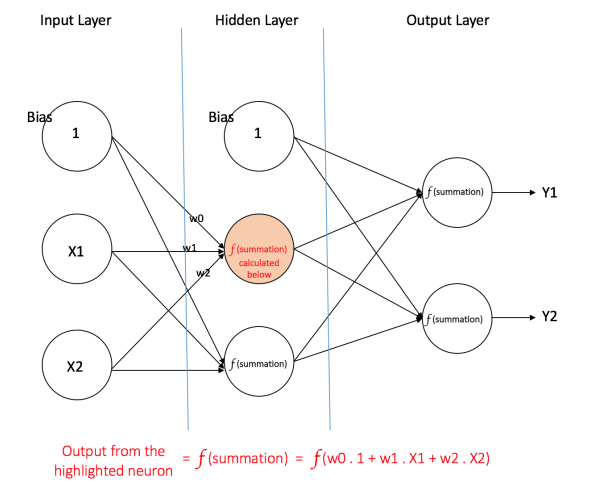
在前馈网络中，信息只向一个方向移动，从输入节点，通过隐含节点（如果有的话）到输出节点，网络中没有循环。

前馈网络有两个例子：

单层感知器：这是最简单的前馈神经网络，不包含任何隐藏层。

多层感知器：多层感知器具有一个或多个隐藏层。在实际应用中，大多使用多层感知机。

多层感知器

多层感知器（MLP）包含一个或多个隐藏层（除了一个输入层和一个输出层）。

上图是具有单个隐含层的多层感知机，输入层有三个节点，偏置节点的值为1，其他两个节点将X1和X2作为外部输入（数值取决于输入数据集）。在输入层中不执行计算，所以来自输入层中的节点的输出分别是1，X1和X2，这些输入被送到隐含层中。

隐含层也有三个节点，偏置节点的输出为1.隐藏层中其他两个节点的输出取决于来自输入层（1，X1，X2）的输出以及与连接（边缘）相关的权重。F是激活函数。经计算后被送到输出层中。

输出层：输出层有两个节点，它们从隐含层获取输入并执行类似的计算，计算的结果（Y1和Y2）就是多层感知器的输出。

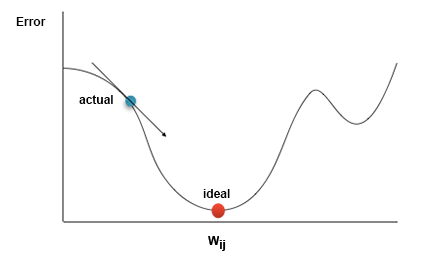
反向传播算法

多层感知器学习的过程称为反向传播算法。通常缩写为BackProp是训练人工神经网络（ANN）的几种方法之一。它是一个有监督的训练。BackProp就像“ 从错误中学习 ”一样，在出现错误时会对其进行更正。

ANN由不同层次的节点组成; 输入层，中间隐含层和输出层。相邻层的节点之间的连接具有与其相关的“权重”。训练的目标是给各个节点分配正确的权重。给定一个输入向量，这些权重决定输出多大向量。

BackProp算法：

最初所有的权重是随机分配的。在给网络一个输入之后，它会产生一个输出，下一步是教网络该输入应该是什么样的正确输出（理想输出）。网络将采用这个理想的输出，并开始调整权重以在下一次产生更准确的输出，从输出层开始并向后返回，直到到达输入层。因此，下一次我们向网络展示相同的输入时，它将使输出更接近我们训练输出的理想输出。这个过程重复许多次迭代，直到认为理想输出和网络输出之间的误差足够小，即到达最小误差。

该算法使用梯度下降计算来调整权重。假设网络输出中的某个权重与错误之间的关系如下图：

梯度下降是在函数空间中寻找一个下降路径，从高处逐步下降，最终找到空间的最低点的过程。算法从起始点开始，逐步向下探索，每一步会求出当前点的导数，进而得知所处位置的“坡度”。接着，算法会选择目前向下坡度最大的一个方向，继续前进。如此以来，梯度下降算法能够找到一个极值点，得到目标函数的极小值。

超参数是指那些需要我们在训练过程中，结合模型的实际表现情况与问题情境，进行调整的参数。梯度下降算法的超参数含有：学习速率a，每次下降使用的样本数量 x，学习速率的衰减率d。

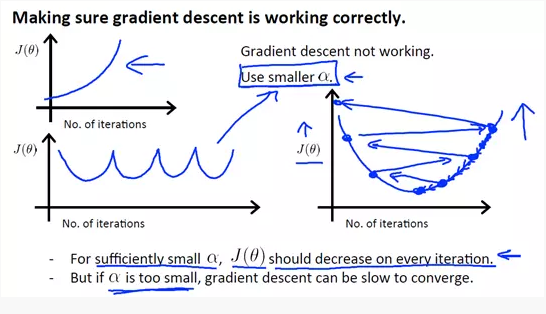
学习速率对于梯度下降算法来说是一个重要的超参数。它代表着算法每一次向下走时的步伐幅度。它的大小决定了算法收敛的速度：a过大时，算法下降速度过快，可能略过终点，而导致在终点处来回徘徊，反而收敛较慢；a过小时，算法下降速度较慢，导致最终收敛时间较长。

根据样本选择的数量，一般分为三类梯度下降算法：

Batch Gradient Descent 批量梯度下降： 这就是普通的梯度下降算法，即整个数据集对于下降的梯度都有贡献。

Stochastic Gradient Descent 随机梯度下降： 每次仅随机地取一组数据作为样本来梯度下降，速度很快，问题在于样本数量小，下降方向不准确，容易陷入局部最优解中。

Mini-Batch Gradient Descent 小批量梯度下降： 即取一部分数据来调整梯度下降算法，它的优点是综合了批量梯度下降和随机梯度下降，既让一定量的数据保证了梯度下降的可靠性，也减小了每回合训练的数据量，加速训练速度。

随着算法逐渐接近终点，容易出现的情况是算法由于步幅过大，导致在终点周围来回徘徊，不易收敛。而若将学习速率调小，则前期可能面临下降速度过慢的问题，导致浪费很多时间。

在训练出模型之后，我们需要通过数据来验证模型的准确率。一般来说，数据集需要分为训练集/验证集/测试集（建议比例6:2:2）或训练集/测试集(7:3)。

交叉验证 Cross Validation

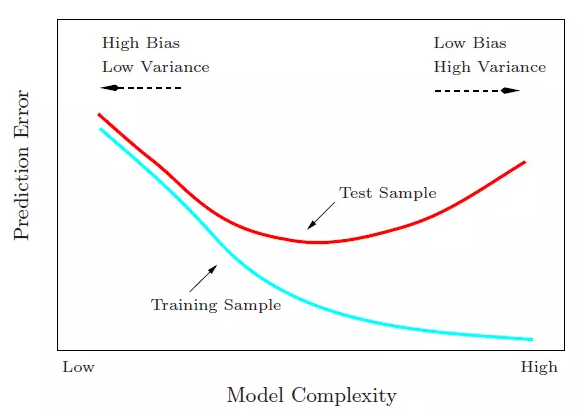
当数据量不够的时候，我们会需要通过交叉验证的方法在保证训练数据量足够的同时来实现合理的验证。在《统计学习方法》中，提到了三种常见的方法：

简单交叉认证： 直接将数据集分为7:3，来进行验证

S折交叉验证(S-fold Cross Validation)： 随机将已给数据切分为S个互不相交的大小相同的子集，用S-1个子集做训练，1个做测试

留一测试： 第二种方法中当 S=N 时的特殊情况

过度拟合（overfitting）： 基于数据的训练，容易使得模型仅在本数据集中产生良好的表现，而在其它的数据集中表现极差，这被称为过拟合现象。



可以看到，一般的机器学习模型来说，随着模型复杂度的上升，测试集与训练集的误差会越来越大，所以我们需要找到一个平衡点，使得模型的误差较小的同时，两个数据集的相对误差也较少。奥卡姆剃刀原理在这里可以有很好的指导：我们需要的是一个能够很好解释已知数据同时最为简单的模型。

在做这个识别字符前，我想到了两种方法来识别，一种是使用HOG和SVM来识别，另一种是使用ANN识别，这两种方法各有优缺点。

我最先做了SVM版的识别，实现了一个简单的光学字符识别（OCR）算法，将训练SVM根据其计算的面向方向梯度（HOG）特征向量来识别手写的小写字母。

1. 初始化HOGDescriptor。HOG特征描述符是用于对象检测的通用描述符，其最初被提出用于行人检测。，HOG使用渐变直方图。梯度由不同角度的边缘检测确定。对于我们的每个字母图像，我们将计算HOG特征矢量，并且我们将向SVM输入这些特征矢量以标示每个字母的特征。
2. 初始化SVM,支持向量机是机器学习为我们提供的各种分类器之一，例如决策树，回归，贝叶斯分类器或神经网络。
3. 训练SVM。需要确保字母在图像的正中间，因为左上角中字母的HOG描述符与右下角角中相同字母的描述符不同。这是因为梯度位于特征向量的不同单元格中。

SVM就是一种二分类器，特征空间上的间隔最大化的分类器，目的是从特征学习中取出一个0/1分类模型，对于二分类问题y只取两个值，目标是求一个超平面。若是遇到非线性的问题，这就要引入核函数，核函数就是把非线性的问题映射为线性的，通过引入核函数，可将隐式得到的非线性数据映射到高维空间中，而不增加参数的个数。建立非线性学习器的2个步骤是：1.使用非线性映射，将数据映射到F。2.在特征空间中使用线性分类器。核函数可以分为这么几类，多项式核函数，斯核函数，线性核函数。他们的本质是遇到高维不可分的情况，可以映射到高维空间，解决映射中的维数爆炸，价值就在他虽然是低维转换到了高维，但是计算还是低维的。

SVM的优缺点：

非线性映射理论基础，利用核函数代替了高维空间的映射，最大化间隔是核心，支持向量是训练的结果，最终结果是少量的向量决定的，可以提出较大的样本，所以有较小的鲁棒性。

不足：对大规模训练难以实施，解决多分类有很大的困难，依赖核函数形式，

ANN的优缺点：

可实现非线性映射，有自学能力，有推广概括能力，直接实现多类分类

不足时采用梯度下降法，速度慢，有可能进入局部最小值而训练失败，新加入的样本有影响，可能会出现欠学习或过学习，依赖隐层等结构设计，依赖激活函数形式

在试过SVM后，觉得不是特别好，虽然只需要少量样本就能实现不错的识别效果，但还是人工神经网络对我更有吸引力。

下面将分别讲解这三大部分的实现过程：

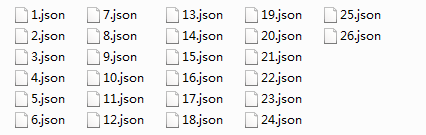
1. 模型部分
2. 准备图片库

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，每张图片都是长28像素宽28像素，这些图片是由不同人手写的0到9的数字。可是我要做的不是识别数字，而是英文字母。我找到了一个和mnist相比不那么有名的数据集：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。emnist里是一系列压缩文件，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。

// digitLoader.js labelsLoader.js rawMaker.js rawWriter.js emnist\_dl.js



二进制的数据不能直接拿来用，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，将二进制数据转换成26个可被js直接读取的json文件，每个文件分别对应一个字母。

这时数据虽然能被读取到，可是不能直接使用，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。

1. 构建网络

神经网络的程序一般是用MATLAB、Python或是C、R等语言实现，但js也有相关的npm包，如Brain.js，ConvNetJS，synaptic，Deeplearnjs等。

这里用到了一个叫Synaptic的npm包，它是一个用于node.js和浏览器的javascript神经网络库，它的内部算法没有其他依赖，所以可以构建和训练基本上任何类型的一阶甚至二阶神经网络。

首先，必须确定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，我设置了1层隐含层，有100个神经元。

另外，还有一些其他参数需要设置

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

学习率决定了参数移动到最优值的速度快慢，如果学习率过大，很可能会越过最优值导致函数无法收敛，甚至发散；反之，如果学习率过小，优化的效率可能过低，算法长时间无法收敛，也易使算法陷入局部最优（非凸函数不能保证达到全局最优）。合适的学习率应该是在保证收敛的前提下，能尽快收敛。

设置较好的learning rate，需要不断尝试。在一开始的时候，可以将其设大一点，这样可以使weights快一点发生改变，在迭代一定的epochs之后人工减小学习率。经过不断尝试，学习率设为0.1时，效果最好。

随着迭代次数的增加，模型在训练集上的误差会越来越小，但也有可能在达到最小误差后不降反升。由于本次训练数据比较大，有两千多张图片，最大迭代次数设大一点比较好。

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。

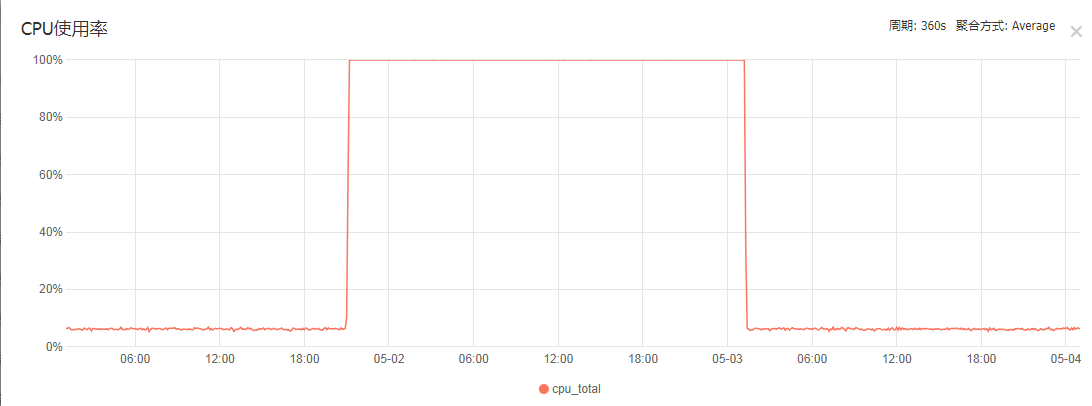
为了让程度能在服务器上跑，这里需要用到守护进程，我选择了forever。forever的用途就是帮我们更好的管理我们node App服务，本质上就是在forever进程之下，创建一个node app的子进程。

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

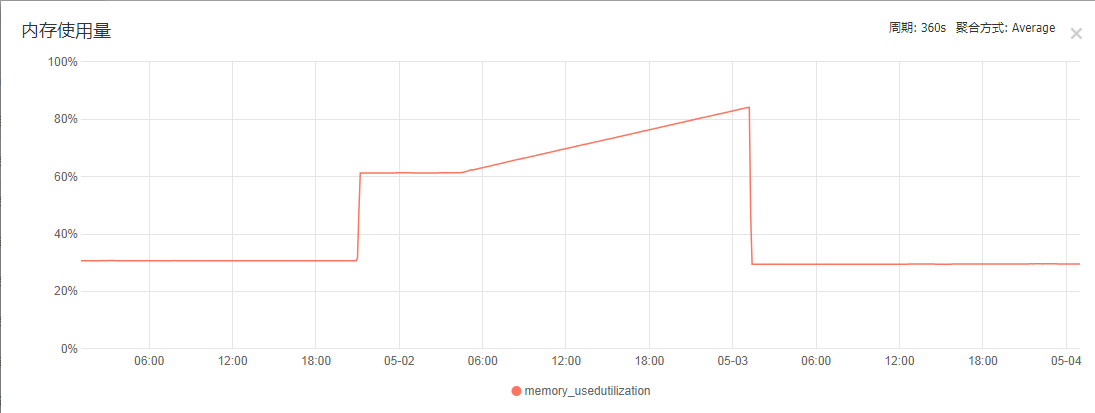
意思是开始执行app-train.js文件，输出日志到train.log中，最大执行次数为1次。

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：







1. 测试模型

训练完成的模型也是一个js文件，6M左右，里面是大量的参数。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，则模型越好。

1. 后端部分
2. 搭建服务

后端使用nodejs，nodejs是运行在服务器上的JavaScript，语法和js基本一样，它是一个基于 Chrome V8 引擎的 JavaScript运行环境，它使用了非阻塞式、事件驱动的I/O模型，轻量又高效。 Node.js 的包管理器 npm，是全球最大的开源库生态系统，每天都有成千上万的人在上面分享自己的开源项目，nodejs之所以能快速的发展，除了天生异步的语言特点外，还有一个重要的原因就是npm。

为了搭建web服务，这里使用了Express，一个基于nodejs的开发框架，

1. 对接模型

由于模型是一个js文件，光是读取并没有什么用处，还需要执行里面的代码，这里需要使用eval语句，返回值是一个函数，将其存起来，使用的时候传入1x784的数组，会返回1x26的识别结果。

1. 写接口

接口是连接前端和模型的桥梁，数据的流通方向大致可描述成：前端-接口-模型-接口-前端。

由于前端传进来的是长宽28像素的二进制图像数据，不同于一般请求，所以需要使用multiparty这个npm包来获取图像数据，而把二进制图像数据转化成数组则需要另一个关键的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。

为了防止前端图像尺寸传错，首先需要对图像resize为28x28。

然后进行图像的预处理，前端返回的图像是256色的位图的二进制数据，由于256色比较复杂，也没有必要，所以需要先进性灰度化处理。灰度化就是图像的每个像素的RGB三个值是相等的，普通彩图的RGB三位的值基本不同，所以会显示出不同的颜色，经过灰度化之后，图像便没有了颜色的差异，只有亮度上的不同。灰度值越大，越亮，最大为255，白色；灰度值越小，则越暗，最小为0，黑色。

注意，这里有一个大坑。将emnist的数据转化为28x28，再画出来后，会发现图不是正的，而是先沿y轴做了一个镜像翻转，再顺时针旋转了90度，所以之后对输入数据也需要进行同样的操作。否则出来的结果千奇百怪。

现在的图像经过灰度化之后还是28x28的二维数组，值是0-255的数字，0代表黑色，255代表白色。在转为一维数组的同时，还需要对里面的灰度值进行计算，即进行二值化。因为原始数据是0-1，0代表白色，1代表黑色。因此需要将0-255转化为1-0，不光大小要归一，值也要反过来。这时需要进行阈值的选择，以阈值为分界线，两边的值为0或1。

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，每项的值代表可能性，越接近1，则越可能是对应的字母。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.

1. HTTPS配置

配置HTTPS需要https证书，而在申请证书之前，需要申请域名，并做好备案，域名指向和解析。

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。

由于https的端口是443，所以本应用的后端只开放443端口。

1. 前端部分

前端部分需要实现的主要功能是用户能手写字母，并将图像发送给后端。这里使用了html中的canvas标签，它是为了绘制客户端矢量图形而设计的。它自己没有行为，但却把一个绘图 API 展现给客户端 JavaScript 以使脚本能够把想绘制的东西都绘制到一块画布上。

<canvas> 标记由 Apple 在 Safari 1.3 Web 浏览器中引入。对 HTML 的这一根本扩展的原因在于，HTML 在 Safari 中的绘图能力也为 Mac OS X 桌面的 Dashboard 组件所使用，并且 Apple 希望有一种方式在 Dashboard 中支持脚本化的图形。

Canvas标签只是图形的容器，具体的绘制还需要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。然后绑定按下事件，当鼠标点击或手指按下时，获取点击点的坐标。再绑定移动事件，当鼠标或手指按住移动时，会频繁触发该事件，将此时的点和上一个点连起来，就绘制出了轨迹。最后绑定抬起事件，当鼠标或手指松开时，轨迹中断，结束绘制。这样，就可以绘制图形了。

另外，绘制完成后需要向后端发送数据，为了尽可能的减少数据量，也为了方便后端处理，在发送前需要将图像调整大小为28x28像素。

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。另外微信小程序规定所有后端接口都需要https加密。

微信小程序

小程序开发框架的目标是通过尽可能简单、高效的方式让开发者可以在微信中开发具有原生 APP 体验的服务。

框架提供了自己的视图层描述语言 WXML 和 WXSS，以及基于 JavaScript 的逻辑层框架，并在视图层与逻辑层间提供了数据传输和事件系统，可以让开发者可以方便的聚焦于数据与逻辑上。框架的核心是一个响应的数据绑定系统。

整个系统分为两块视图层（View）和逻辑层（App Service）框架可以让数据与视图非常简单地保持同步。当做数据修改的时候，只需要在逻辑层修改数据，视图层就会做相应的更新。

作为一个前端开发者，刚接触小程序时，会有一种非常熟悉的感觉，因为它的响应式的数据绑定和现在流行的前端框架React.js、Vue.js等实现的效果几乎一模一样，所以上手非常快。

小程序使用了前端技术栈 JavaScript/WXML/WXSS。但和常规的前端开发又有一些区别。

JavaScript: 微信小程序的 JavaScript 运行环境即不是 Browser 也不是 Node.js。它运行在微信 App 的上下文中，不能操作 Browser context 下的 DOM，也不能通过 Node.js 相关接口访问操作系统 API。所以，严格意义来讲，微信小程序并不是 Html5，虽然开发过程和用到的技术栈和 Html5 是相通的。

WXML: 作为微信小程序的展示层，并不是使用 Html，而是自己发明的基于 XML 语法的描述。

WXSS: 用来修饰展示层的样式。WXSS (WeiXin Style Sheets) 是一套样式语言，用于描述 WXML 的组件样式。WXSS 用来决定 WXML 的组件应该怎么显示。

微信小程序最大的好处是不需要做设备适配，只要微信能运行，小程序就能运行。小程序虽然是一个封闭形态下的前端开发技术，但借助微信的巨大影响力，越来越流行。

https

https是具有安全性的ssl加密传输协议，不同于http的80端口，它使用的是443端口，是由SSL+HTTP协议建立的可进行身份认证、加密传输的网络协议，比http协议更安全，但由于需要进行加密解密，也比http慢一点，不过和安全性相比，这点速度的牺牲是完全有必要的。

附录：

1.Emnist：

// 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 / 10 | 0;

var test = \_test / 10 | 0;

if (training < 1)

training = 1;

if (test < 1)

test = 1;

if (training + test + 1 > MNIST.\_\_MINLENGTH) {

console.warn('There are not enough samples to make a training set of ' + training + ' elements and a test set of ' + test + ' elements.');

if (training > test) {

test = MNIST.\_\_MINLENGTH \* (test / training);

training = MNIST.\_\_MINLENGTH - training;

}

else {

training = MNIST.\_\_MINLENGTH \* (training / test);

test = MNIST.\_\_MINLENGTH - test;

}

}

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;

}

2.app-train

const fs = require('fs')

/\* 准备数据 \*/

const emnist = require('./emnist')

const set = emnist.set(2002, 26) // 2002 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: 0.1, // 学习率

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("./net3.js", standalone, function(err) {

if (err) {

console.log(err)

} else {

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

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

}

});

1. app-train

const http = require('http')

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()

app.use(bodyParse.json());

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

let netData = fs.readFileSync('./nets/net0-03.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);

http.createServer(app).listen(8081, function() {

console.log('listening port 8081')

})

1. digitsLoader

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;

}

1. labelsLoader

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;

}

1. rawMaker

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;

}

1. rawWriter

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();

}

};

1. 前端核心代码

<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>

//logs.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();

},

// 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: 'http://192.168.199.219:8081/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)

}

}

})