基于人工神经网络的在线识别Web应用

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

基础知识

神经网络的基础单元是神经元。神经元就像是一个函数，又称激活函数，输入数据，然后给出输出结果。神经元有很多类型，我使用sigmoid神经元，亦即一个特定的函数，此函数可以将x值映射到0和1之间。

输入乘以权重就是神经元的总输入X，然后总输入经过sigmoid函数后，得到输出Y，即sigmoid(X)=Y，这样，就将输入映射到了0和1之间。神经元通过突触相互连接，从输入到输出，逐层相连后即可形成神经网络。

突触的权重一般是先给定一个初始值，然后开始正向进行神经元计算，一直到输出结果，然后根据误差，开始反向传播，进行误差调整和偏差值调整。因为

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。

// emnist.js

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等语言实现，但作为一个js的忠实拥护者，我认为其他语言能做的，js一样能做，甚至某些方面能做得更好，所以我决定用js来实现模型的制作。

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：0.2

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。

1. 测试模型

训练完成的模型也是一个js文件，5M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。然后将图像灰度化。

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。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。

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. 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。

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。https的申请大部分是需要收费的，不过万幸的是阿里云提供免费的证书。