# 使用tensorflow实现mnist手写识别

## 1.mnist简介

这个MNIST数据库是一个手写数字的数据库，它提供了六万的训练集和一万的测试集。它的图片是被规范处理过的，是一张被放在中间部位的28px\*28px的灰度图。

总共4个文件:



图片都被转成二进制放到了文件里面，所以，每一个文件头部几个字节都记录着这些图片的信息，然后才是储存的图片信息。

标记文件(label)二进制信息：

**[**offset**]** **[**type**]** **[**value**]** **[**description**]**

0000 32 bit integer 0x00000801**(**2049**)** magic number **(**MSB first**)**

0004 32 bit integer 60000 number of items

0008 unsigned byte **??** label

0009 unsigned byte **??** label

**........**

xxxx unsigned byte **??** label

The labels values are 0 to 9.

图片文件(image)二进制信息：

**[**offset**]** **[**type**]** **[**value**]** **[**description**]**

0000 32 bit integer 0x00000803**(**2051**)** magic number

0004 32 bit integer 60000 number of images

0008 32 bit integer 28 number of rows

0012 32 bit integer 28 number of columns

0016 unsigned byte **??** pixel

0017 unsigned byte **??** pixel

**........**

xxxx unsigned byte **??** pixel

每个像素被转成了0**-**255**,**0代表着白色，255代表着黑色。

mnist数据可以通过如下代码直接下载：

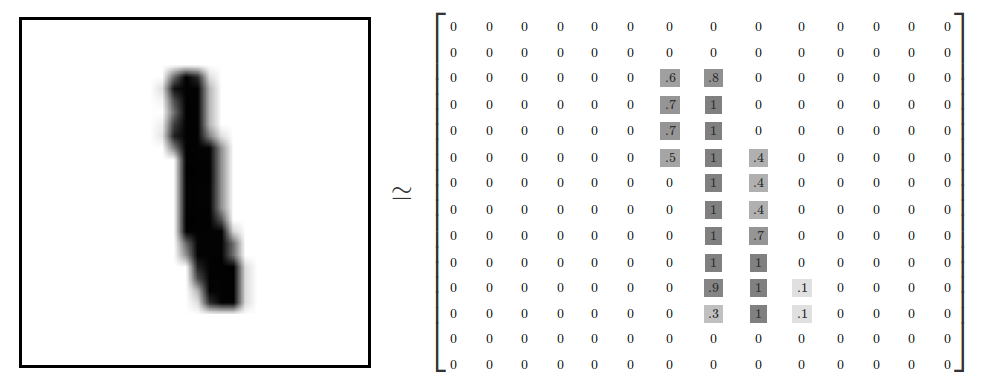
from tensorflow**.**examples**.**tutorials**.**mnist import input\_data

mnist **=** input\_data**.**read\_data\_sets**(**"MNIST\_data/"**,** one\_hot**=**True**)**

下载下来的数据集被分成两部分：60000行的训练数据集（mnist.train）和10000行的测试数据集（mnist.test）。这样的切分很重要，在机器学习模型设计时必须有一个单独的测试数据集不用于训练而是用来评估这个模型的性能，从而更加容易把设计的模型推广到其他数据集上（泛化）。

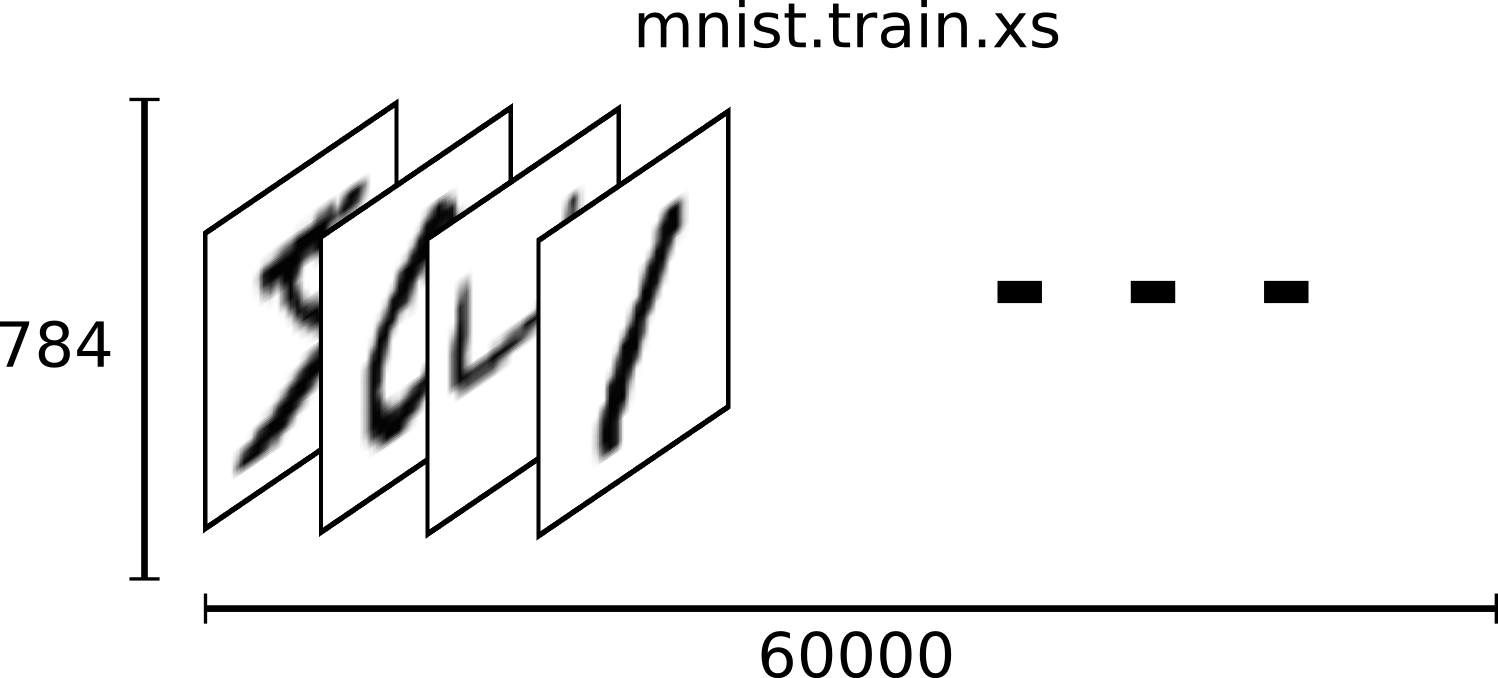
正如前面提到的一样，每一个MNIST数据单元有两部分组成：一张包含手写数字的图片和一个对应的标签。我们把这些图片设为“xs”，把这些标签设为“ys”。训练数据集和测试数据集都包含xs和ys，比如训练数据集的图片是 mnist.train.images ，训练数据集的标签是 mnist.train.labels。

每一张图片包含28X28个像素点。我们可以用一个数字数组来表示这张图片：

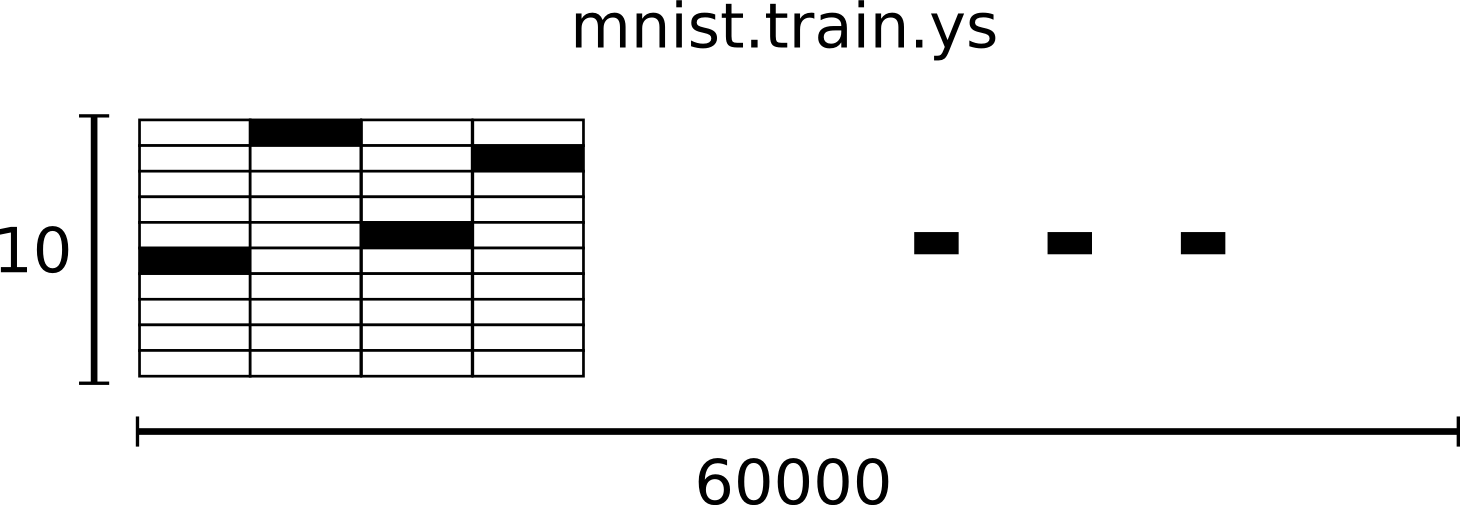


我们把这个数组展开成一个向量，长度是 28x28 = 784。

因此，在MNIST训练数据集中，mnist.train.images 是一个形状为 [60000, 784] 的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。在此张量里的每一个元素，都表示某张图片里的某个像素的强度值，值介于0和1之间。



相对应的MNIST数据集的标签是介于0到9的数字，用来描述给定图片里表示的数字。为了用于这个教程，我们使标签数据是"one-hot vectors"。 一个one-hot向量除了某一位的数字是1以外其余各维度数字都是0。所以在此教程中，数字n将表示成一个只有在第n维度（从0开始）数字为1的10维向量。比如，标签0将表示成([1,0,0,0,0,0,0,0,0,0,0])。因此， mnist.train.labels 是一个 [60000, 10] 的数字矩阵。



## 2.一些数学知识

### 2.1 softmax

softmax可以理解为归一化，如目前图片分类有一百种，那经过 softmax 层的输出就是一个一百维的向量。向量中的第一个值就是当前图片属于第一类的概率值，向量中的第二个值就是当前图片属于第二类的概率值…这一百维的向量之和为1。

所以0~9手写识别变成了实现这样一个分类器： 给定一张手写图片，分类器给出改数字属于哪个分类。（0-9共10个分类）

我们先来直观看一下，Softmax究竟是什么意思？

我们知道max，假如说我有两个数，a和b，并且a>b，如果取max，那么就直接取a，没有第二种可能。但有的时候我不想这样，因为这样会造成分值小的那个饥饿。所以我希望分值大的那一项经常取到，分值小的那一项也偶尔可以取到，那么我用softmax就可以了。

现在还是a和b，a>b，如果我们取按照softmax来计算取a和b的概率，那a的softmax值大于b的，所以a会经常取到，而b也会偶尔取到，概率跟它们本来的大小有关。

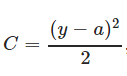
**softmax定义：**

假设我们有一个数组，V，Vi表示V中的第i个元素，那么这个元素的Softmax值就是：

也就是说，是该元素的指数，与所有元素指数和的比值。

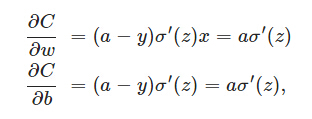
### 2.2 方差代价函数

代价函数经常用方差代价函数（即采用均方误差MSE），比如对于一个神经元（单输入单输出，sigmoid函数）,定义其代价函数为：



其中y是我们期望的输出，a为神经元的实际输出【 a=σ(z), where z=wx+b 】。

在训练神经网络过程中，我们通过梯度下降算法来更新w和b，因此需要计算代价函数对w和b的导数：

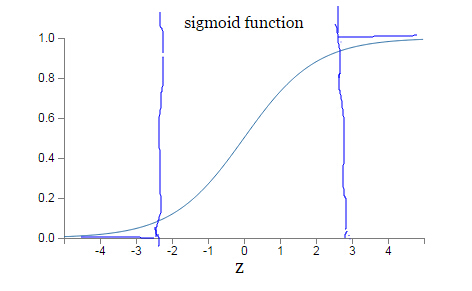


然后更新w、b：

w <—— w - η\* ∂C/∂w = w - η \* a \*σ′(z)

b <—— b - η\* ∂C/∂b = b - η \* a \* σ′(z)

因为sigmoid函数的性质，导致σ′(z)在z取大部分值时会很小（如下图标出来的两端，几近于平坦），这样会使得w和b更新非常慢（因为η \* a \* σ′(z)这一项接近于0）。



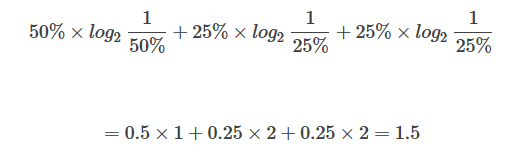
### 2.3 交叉熵代价函数

为了克服这个缺点，引入了交叉熵代价函数。

熵在信息论里面衡量的是数据的混乱程度，本质是香农信息量(plog21p,0<p≤1)的期望。

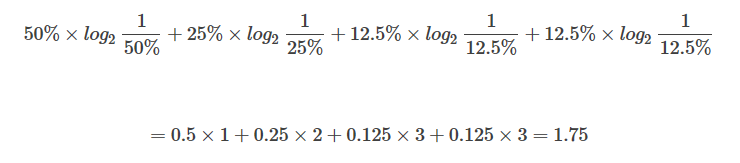
如果直接用纯数学语言来解释，一定晦涩难懂，因此我想到了一个比较有趣的实例给大家理解一下。

熟悉足球的童鞋们应该都记得，在2002年，中国足球队首次也是唯一一次杀入了世界杯，全国球迷无不为之热血沸腾。但当时中国队所在的C组号称死亡之组，组内其它三支球队分别是巴西（最后的冠军）、土耳其（最后的季军）和哥斯达黎加。现在命运之神分配给这四支球队夺得小组第一的概率分别为巴西50%，土耳其25%， 哥斯达黎加25%， 中国0%。我们假定这是真实概率，那么这一组概率的信息量是多大呢？按照香农信息量的计算公式，我们可以得出：



因此，这一组概率的信息量就是1.5。香农认为，信息量的度量等于不确定性的多少，可以用 “比特”（bit）这个概念来度量。比如土耳其夺得小组第一的概率是1/4，我们就需要用一个2位的bit来表示，又因为其只占了4份中的1份，因此它的信息量就是0.25×2=0.5。然后我们在赛前预测各个队夺得小组第一的概率，恰好也给出“巴西50%，土耳其25%， 哥斯达黎加25%， 中国0%”这样的预测。根据前文可知，这组预测的信息量当然也是1.5。

接着交叉熵可以来了，这组预测结果的交叉熵为1.5−1.5=0，表示预测结果与真实结果之间的差异程度为0，即我们的预测百分之百正确。这时02国家队的球迷不干了：凭什么不给咱国家队一丝期待？那好，不妨修改一下我们的赛前预测：巴西50%，土耳其25%， 哥斯达黎加12.5%， 中国12.5%。国家队有八分之一的概率拿小组第一，这期待够带劲吧。再算一下这组预测结果的熵：



交叉熵为1.75=1.5=0.25，大于0，表示预测结果与真实结果之间存在差异，这里的差异便是中国队和哥斯达黎加队的预测结果。不难看出，交叉熵的值越大，这种差异程度也就越大。

**总结一下：**

现有关于样本集的2个概率分布p和q，其中p为真实分布，q非真实分布。按照真实分布p来衡量识别一个样本的所需要的编码长度的期望(即平均编码长度)为：H(p)= 。

如果使用错误分布q来表示来自真实分布p的平均编码长度，则应该是：H(p,q)= 。因为用q来编码的样本来自分布p，所以期望H(p,q)中概率是p(i)。H(p,q)我们称之为“交叉熵”。

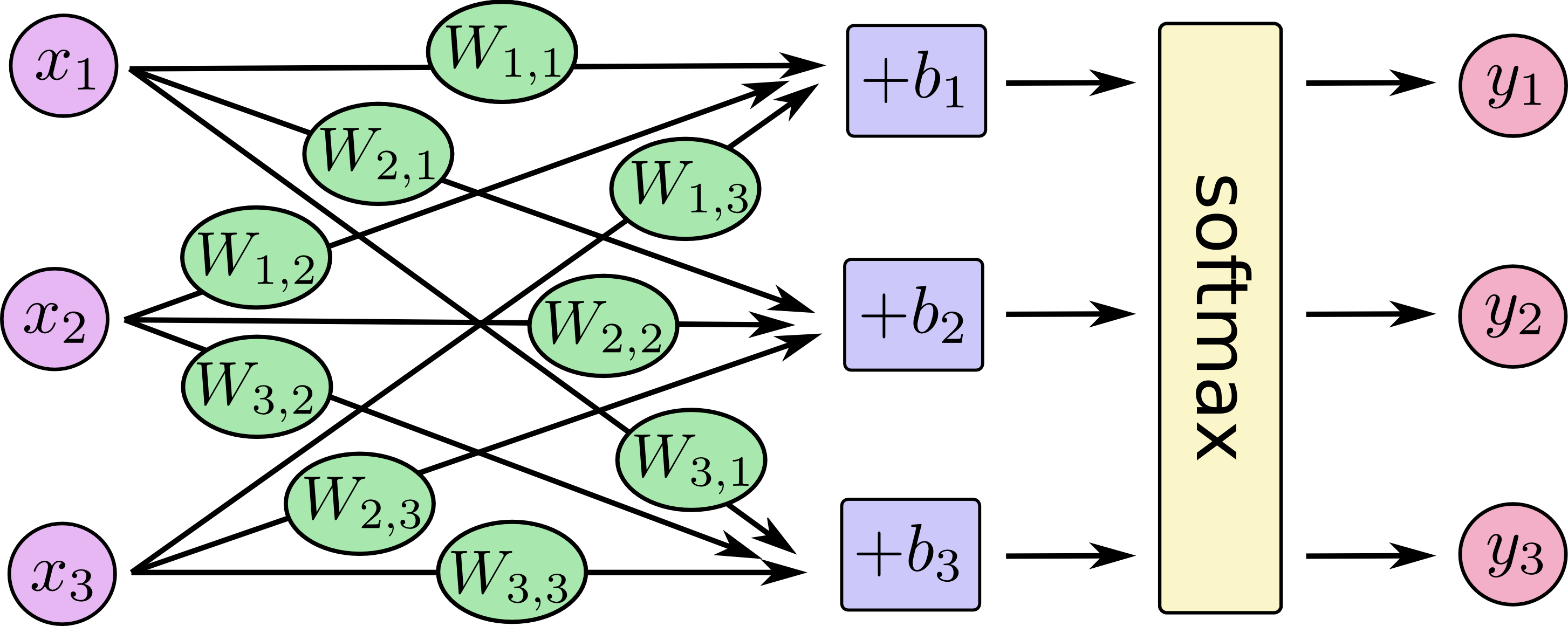
## 3.简单的神经网络实现mnist手写识别

### 3.1 模型设计

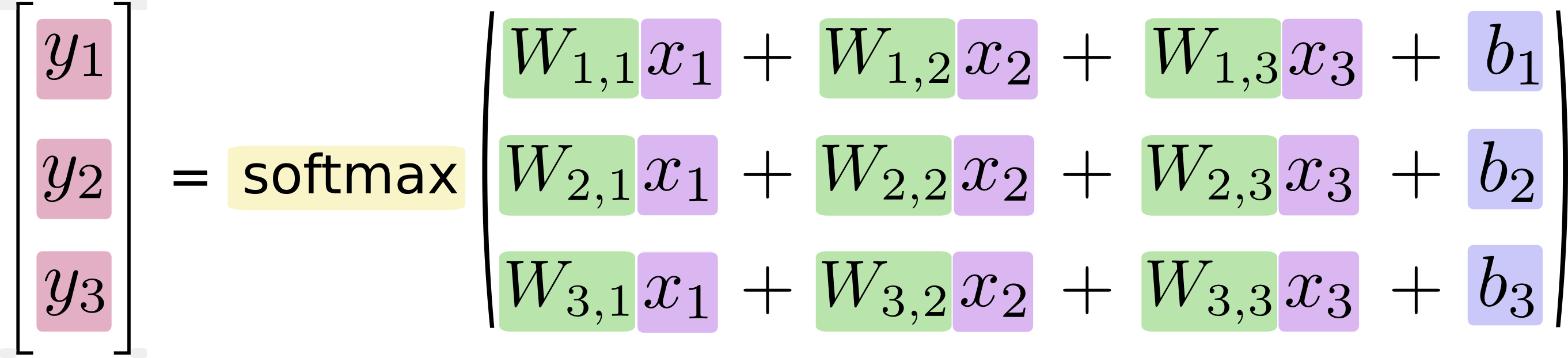
在这个例子中我们使用了softmax回归模型。

#### 3.1.1 softmax回归模型简介

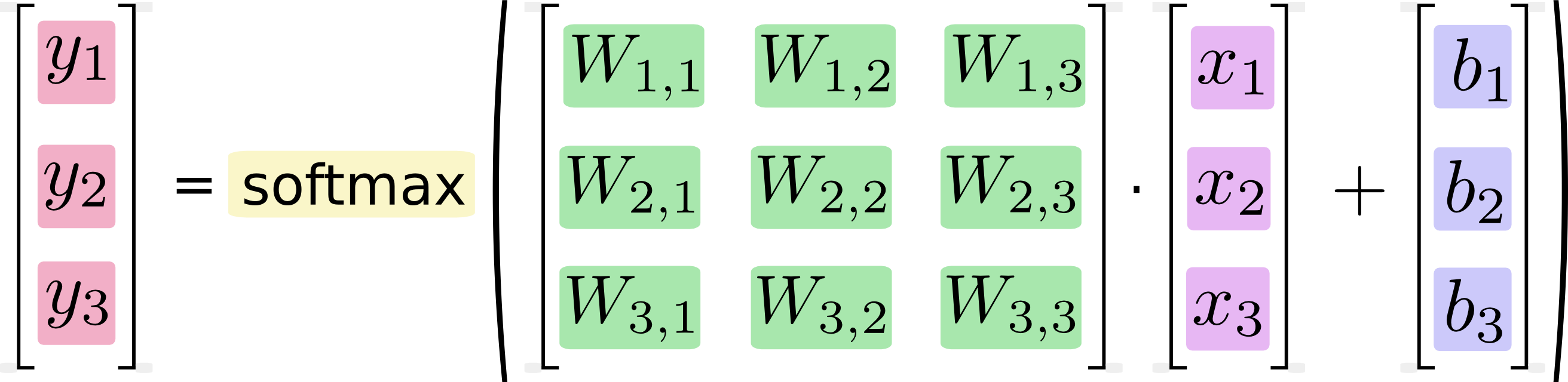
我们知道MNIST的每一张图片都表示一个数字，从0到9。我们希望得到给定图片代表每个数字的概率。比如说，我们的模型可能推测一张包含9的图片代表数字9的概率是80%但是判断它是8的概率是5%（因为8和9都有上半部分的小圆），然后给予它代表其他数字的概率更小的值。



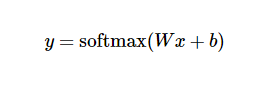
如果把它写成一个等式，我们可以得到：



我们也可以用向量表示这个计算过程：用矩阵乘法和向量相加。这有助于提高计算效率。（也是一种更有效的思考方式）

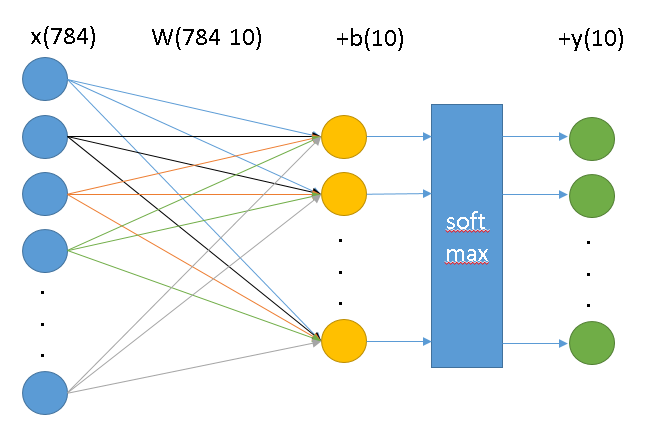


更进一步，可以写成更加紧凑的方式：



#### 3.1.2 模型建立相关代码分析

在本例中softmax回归模型示意图如下：



输入为一张图像包含784个像素，输出为该图像0~9对应的概率。

首先定义输入参数x，x不是一个特定的值，而是一个占位符placeholder，我们在TensorFlow运行计算时输入这个值。我们希望能够输入任意数量的MNIST图像，每一张图展平成784维的向量。我们用2维的浮点数张量来表示这些图，这个张量的形状是[None，784]。

x **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 784**])**

接下来定义权重和偏置。

W **=** tf**.**Variable**(**tf**.**zeros**([**784**,** 10**]))**

b **=** tf**.**Variable**(**tf**.**zeros**([**10**]))**

在这里，我们都用全为零的张量来初始化W和b。因为我们要学习W和b的值，它们的初值可以随意设置。W的维度是[784，10]，因为我们想要用784维的图片向量乘以它以得到一个10维的证据值向量，每一位对应不同数字类。b的形状是[10]，所以我们可以直接把它加到输出上面。

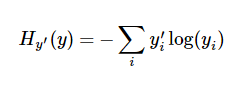
最后便是整个softmax模型的定义了，softmax(y=wx+b)。

y **=** tf**.**nn**.**softmax**(**tf**.**matmul**(**x**,** W**)** **+** b**)**

### 3.2 模型训练

#### 3.2.1 损失函数

为了训练我们的模型，我们首先需要定义一个指标来评估这个模型是好的。其实，在机器学习，我们通常定义指标来表示一个模型是坏的，这个指标称为成本（cost）或损失（loss），然后尽量最小化这个指标。但是，这两种方式是相同的。

这里使用交叉熵作为评估函数，交叉熵的数学定义如下：

为了计算交叉熵，我们首先需要添加一个新的占位符用于输入正确值。

y\_ **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 10**])**

计算交叉熵代码如下：

cross\_entropy **=** **-**tf**.**reduce\_sum**(**y\_**\***tf**.**log**(**y**))**

首先，用 tf.log 计算 y 的每个元素的对数。接下来，我们把 y\_ 的每一个元素和 tf.log(y) 的对应元素相乘。最后，用 tf.reduce\_sum 计算张量的所有元素的总和。注意，这里的交叉熵不仅仅用来衡量单一的一对预测和真实值，而是所有100幅图片的交叉熵的总和。对于100个数据点的预测表现比单一数据点的表现能更好地描述我们的模型的性能。

接下来就要使用tensorflow进行训练了。TensorFlow拥有一张描述你各个计算单元的图，它可以自动地使用反向传播算法(backpropagation algorithm)来有效地确定你的变量是如何影响你想要最小化的那个成本值的。然后，TensorFlow会用你选择的优化算法来不断地修改变量以降低成本。

train\_step **=** tf**.**train**.**GradientDescentOptimizer**(**0.01**).**minimize**(**cross\_entropy**)**

在这里，我们要求TensorFlow用梯度下降算法（gradient descent algorithm）以0.01的学习速率最小化交叉熵。梯度下降算法（gradient descent algorithm）是一个简单的学习过程，TensorFlow只需将每个变量一点点地往使成本不断降低的方向移动。当然TensorFlow也提供了其他许多优化算法：只要简单地调整一行代码就可以使用其他的算法。

TensorFlow在这里实际上所做的是，它会在后台给描述你的计算的那张图里面增加一系列新的计算操作单元用于实现反向传播算法和梯度下降算法。然后，它返回给你的只是一个单一的操作，当运行这个操作时，它用梯度下降算法训练你的模型，微调你的变量，不断减少成本。

#### 3.2.2 开始训练

开始训练之前创建一个session并初始化变量。

sess **=** tf**.**InteractiveSession**()**

tf**.**global\_variables\_initializer**().**run**()**

下面就要开始真正的训练了。

**for** \_ in range**(**1000**):**

batch\_xs**,** batch\_ys **=** mnist**.**train**.**next\_batch**(**100**)**

sess**.**run**(**train\_step**,** feed\_dict**={**x**:** batch\_xs**,** y\_**:** batch\_ys**})**

这里我们让模型循环训练1000次，该循环的每个步骤中，我们都会随机抓取训练数据中的100个批处理数据点，然后我们用这些数据点作为参数替换之前的占位符来运行train\_step。

其中batch\_xs表示100张图片，batch\_ys表示对应的label。

使用一小部分的随机数据来进行训练被称为随机训练（stochastic training）- 在这里更确切的说是随机梯度下降训练。在理想情况下，我们希望用我们所有的数据来进行每一步的训练，因为这能给我们更好的训练结果，但显然这需要很大的计算开销。所以，每一次训练我们可以使用不同的数据子集，这样做既可以减少计算开销，又可以最大化地学习到数据集的总体特性。

### 3.3 模型评估

在mnist数据集中，有一个包含10000个图片样本和标记的数据集，该数据集用于训练模型的评估。

首先让我们找出那些预测正确的标签。

correct\_prediction **=** tf**.**equal**(**tf**.**argmax**(**y**,** 1**),** tf**.**argmax**(**y\_**,** 1**))**

当输入一张图片时，0~9预测对应的概率存放在y中，真实概率存放在y\_中。

tf**.**argmax会找到tensor对象在某一维上的其数据最大值所在的索引值，如果y中的最大值索引和y\_中的最大值索引相同，也就是预测和实际是一样的。

这行代码会给我们一组布尔值。为了确定正确预测项的比例，我们可以把布尔值转换成浮点数，然后取平均值。例如，[True, False, True, True] 会变成 [1,0,1,1] ，取平均值后得到 0.75。

accuracy **=** tf**.**reduce\_mean**(**tf**.**cast**(**correct\_prediction**,** tf**.**float32**))**

最后，我们计算所学习到的模型在测试数据集上面的正确率。

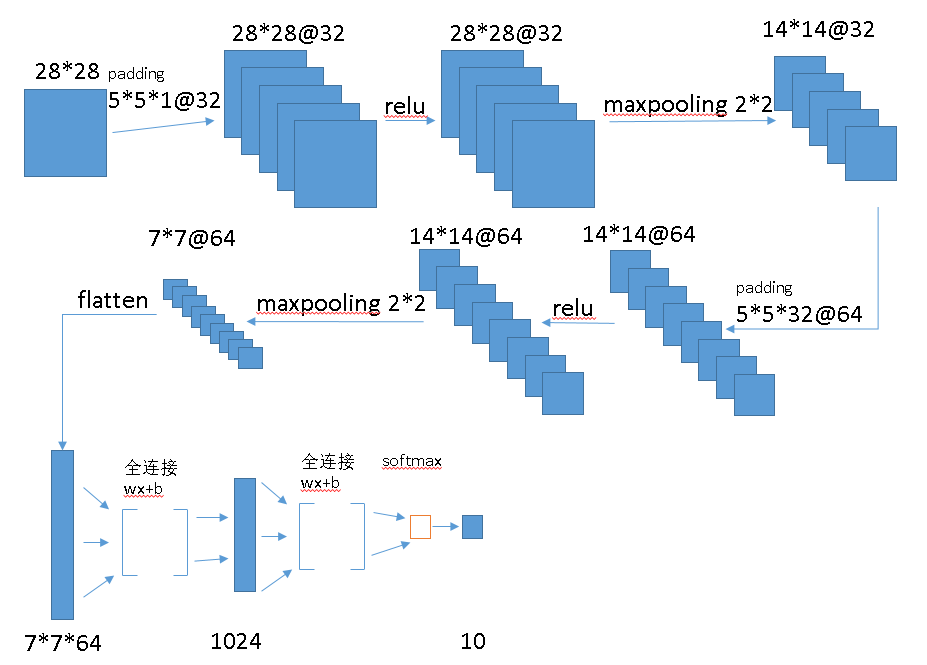
print**(**sess**.**run**(**accuracy**,** feed\_dict**={**x**:** mnist**.**test**.**images**,** y\_**:** mnist**.**test**.**labels**}))**

这个最终结果值应该大约是91%。

## 4.用卷积网络实现mnist手写识别

下面我们将会使用卷积网络和全连接层来改进手写识别的准确度。

下图为该节使用的深度学习网络示意图。



### 4.1 权重初始化

为了创建这个模型，我们需要创建大量的权重和偏置项。这个模型中的权重在初始化时应该加入少量的噪声来打破对称性以及避免0梯度。由于我们使用的是ReLU神经元，因此比较好的做法是用一个较小的正数来初始化偏置项，以避免神经元节点输出恒为0的问题（dead neurons）。为了不在建立模型的时候反复做初始化操作，我们定义两个函数用于初始化。

def weight\_variable**(**shape**):**

initial **=** tf**.**truncated\_normal**(**shape**,** stddev**=**0.1**)**

**return** tf**.**Variable**(**initial**)**

def bias\_variable**(**shape**):**

initial **=** tf**.**constant**(**0.1**,** shape**=**shape**)**

**return** tf**.**Variable**(**initial**)**

下面介绍一下tf.truncated\_normal和tf.constant。

tf**.**truncated\_normal**(**shape**,** mean**,** stddev**)**

shape表示生成张量的维度，mean是均值，stddev是标准差。

这个函数产生正态分布，均值和标准差自己设定。这是一个截断的产生正态分布的函数，就是说产生正态分布的值如果与均值的差值大于两倍的标准差，那就重新生成。和一般的正态分布的产生随机数据比起来，这个函数产生的随机数与均值的差距不会超过两倍的标准差，但是一般的别的函数是可能的。

tf.constant则会产生一个常量矩阵。

### 4.2 卷积和池化

我们的卷积使用1步长（stride size），0边距（padding size）的模板，保证输出和输入是同一个大小。我们的池化用简单传统的2x2大小的模板做max pooling。为了代码更简洁，我们把这部分抽象成一个函数。

def conv2d**(**x**,** W**):**

"""conv2d returns a 2d convolution layer with full stride."""

**return** tf**.**nn**.**conv2d**(**x**,** W**,** strides**=[**1**,** 1**,** 1**,** 1**],** padding**=**'SAME'**)**

def max\_pool\_2x2**(**x**):**

"""max\_pool\_2x2 downsamples a feature map by 2X."""

**return** tf**.**nn**.**max\_pool**(**x**,** ksize**=[**1**,** 2**,** 2**,** 1**],**

strides**=[**1**,** 2**,** 2**,** 1**],** padding**=**'SAME'**)**

### 4.3 网络层

网络层的详细流程请见图。

def deepnn**(**x**):**

#将768的像素转换为28\*28矩阵

with tf**.**name\_scope**(**'reshape'**):**

x\_image **=** tf**.**reshape**(**x**,** **[-**1**,** 28**,** 28**,** 1**])**

#第一层卷积

with tf**.**name\_scope**(**'conv1'**):**

W\_conv1 **=** weight\_variable**([**5**,** 5**,** 1**,** 32**])**

b\_conv1 **=** bias\_variable**([**32**])**

h\_conv1 **=** tf**.**nn**.**relu**(**conv2d**(**x\_image**,** W\_conv1**)** **+** b\_conv1**)**

#最大池化2\*2

with tf**.**name\_scope**(**'pool1'**):**

h\_pool1 **=** max\_pool\_2x2**(**h\_conv1**)**

#第二层卷积

with tf**.**name\_scope**(**'conv2'**):**

W\_conv2 **=** weight\_variable**([**5**,** 5**,** 32**,** 64**])**

b\_conv2 **=** bias\_variable**([**64**])**

h\_conv2 **=** tf**.**nn**.**relu**(**conv2d**(**h\_pool1**,** W\_conv2**)** **+** b\_conv2**)**

#最大池化2\*2

with tf**.**name\_scope**(**'pool2'**):**

h\_pool2 **=** max\_pool\_2x2**(**h\_conv2**)**

#第一个全连接层

with tf**.**name\_scope**(**'fc1'**):**

W\_fc1 **=** weight\_variable**([**7 **\*** 7 **\*** 64**,** 1024**])**

b\_fc1 **=** bias\_variable**([**1024**])**

h\_pool2\_flat **=** tf**.**reshape**(**h\_pool2**,** **[-**1**,** 7**\***7**\***64**])**

h\_fc1 **=** tf**.**nn**.**relu**(**tf**.**matmul**(**h\_pool2\_flat**,** W\_fc1**)** **+** b\_fc1**)**

#定义dropout层，防止过拟合

with tf**.**name\_scope**(**'dropout'**):**

keep\_prob **=** tf**.**placeholder**(**tf**.**float32**)**

h\_fc1\_drop **=** tf**.**nn**.**dropout**(**h\_fc1**,** keep\_prob**)**

#第二个全连接层

with tf**.**name\_scope**(**'fc2'**):**

W\_fc2 **=** weight\_variable**([**1024**,** 10**])**

b\_fc2 **=** bias\_variable**([**10**])**

y\_conv **=** tf**.**nn**.**softmax**(**tf**.**matmul**(**h\_fc1\_drop**,** W\_fc2**)** **+** b\_fc2**)**

**return** y\_conv**,** keep\_prob

### 4.4 模型训练与评估

模型的训练与评估和3中的实验类似

def main**(**\_**):**

#读取mnist数据集

mnist **=** input\_data**.**read\_data\_sets**(**FLAGS**.**data\_dir**,** one\_hot**=**True**)**

#图片输入

x **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 784**])**

#图片识别正确的结果

y\_ **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 10**])**

#构建神经网络

y\_conv**,** keep\_prob **=** deepnn**(**x**)**

#定义损失函数 交叉熵

with tf**.**name\_scope**(**'loss'**):**

cross\_entropy **=** **-**tf**.**reduce\_sum**(**y\_**\***tf**.**log**(**y\_conv**))**

#定义ADAM优化器来做梯度最速下降，

with tf**.**name\_scope**(**'adam\_optimizer'**):**

train\_step **=** tf**.**train**.**AdamOptimizer**(**1e-4**).**minimize**(**cross\_entropy**)**

#是否预测成功的标志

with tf**.**name\_scope**(**'accuracy'**):**

correct\_prediction **=** tf**.**equal**(**tf**.**argmax**(**y\_conv**,** 1**),** tf**.**argmax**(**y\_**,** 1**))**

correct\_prediction **=** tf**.**cast**(**correct\_prediction**,** tf**.**float32**)**

accuracy **=** tf**.**reduce\_mean**(**correct\_prediction**)**

#开始训练和预测

with tf**.**Session**()** as sess**:**

sess**.**run**(**tf**.**global\_variables\_initializer**())**

**for** i in range**(**20000**):**

batch **=** mnist**.**train**.**next\_batch**(**50**)**

**if** i **%** 100 **==** 0**:**

train\_accuracy **=** accuracy**.**eval**(**feed\_dict**={**x**:** batch**[**0**],** y\_**:** batch**[**1**],** keep\_prob**:** 1.0**})**

print**(**'step %d, training accuracy %g' **%** **(**i**,** train\_accuracy**))**

train\_step**.**run**(**feed\_dict**={**x**:** batch**[**0**],** y\_**:** batch**[**1**],** keep\_prob**:** 0.5**})**

print**(**'test accuracy %g' **%** accuracy**.**eval**(**feed\_dict**={**x**:** mnist**.**test**.**images**,** y\_**:** mnist**.**test**.**labels**,** keep\_prob**:** 1.0**}))**

最终可以达到99.2%的成功率。

## 5. 保存训练结果用于手写识别

我们可以将训练好的模型保存起来，方便使用。当输入一张手写图片，可以使用训练过的模型输出识别结果。

### 5.1 训练模型保存

模型保存非常简单，只需再训练后执行下面的代码。

saver **=** tf**.**train**.**Saver**()**

save\_path **=** saver**.**save**(**sess**,** "model.ckpt"**)**

### 5.2 图片预处理

给定一张图片，需要将其转换为28\*28大小，然后变成784大小的一维矩阵。

def imageprepare**(**argv**):**

"""

This function returns the pixel values**.**

The imput is a png file location**.**

"""

im **=** Image**.**open**(**argv**).**convert**(**'L'**)**

width **=** float**(**im**.**size**[**0**])**

height **=** float**(**im**.**size**[**1**])**

newImage **=** Image**.new(**'L'**,** **(**28**,** 28**),** **(**255**))** #creates white canvas of 28x28 pixels

**if** width **>** height**:** #check which dimension is bigger

#Width is bigger. Width becomes 20 pixels.

nheight **=** int**(**round**((**20.0**/**width**\***height**),**0**))** #resize height according to ratio width

**if** **(**nheight **==** 0**):** #rare **case** but minimum is 1 pixel

nheight **=** 1

# resize and sharpen

img **=** im**.**resize**((**20**,**nheight**),** Image**.**ANTIALIAS**).**filter**(**ImageFilter**.**SHARPEN**)**

wtop **=** int**(**round**(((**28 **-** nheight**)/**2**),**0**))** #caculate horizontal pozition

newImage**.**paste**(**img**,** **(**4**,** wtop**))** #paste resized image on white canvas

**else:**

#Height is bigger. Heigth becomes 20 pixels.

nwidth **=** int**(**round**((**20.0**/**height**\***width**),**0**))** #resize width according to ratio height

**if** **(**nwidth **==** 0**):** #rare **case** but minimum is 1 pixel

nwidth **=** 1

# resize and sharpen

img **=** im**.**resize**((**nwidth**,**20**),** Image**.**ANTIALIAS**).**filter**(**ImageFilter**.**SHARPEN**)**

wleft **=** int**(**round**(((**28 **-** nwidth**)/**2**),**0**))** #caculate vertical pozition

newImage**.**paste**(**img**,** **(**wleft**,** 4**))** #paste resized image on white canvas

newImage**.**save**(**"sample.png"**)**

tv **=** list**(**newImage**.**getdata**())** #get pixel values

#normalize pixels to 0 and 1. 0 is pure white, 1 is pure black.

tva **=** **[** **(**255**-**x**)\***1.0**/**255.0 **for** x in tv**]**

#print(tva)

**return** tva

### 5.3 图片识别

当获取到图片对应的一维数组，需要将其输入到先前定义好的网络中去，并导入训练过的模型文件，最后进行计算。

#### 5.3.1 简单神经网络的识别函数

def predictint**(**imvalue**):**

# Define the model (same as when creating the model file)

x **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 784**])**

W **=** tf**.**Variable**(**tf**.**zeros**([**784**,** 10**]))**

b **=** tf**.**Variable**(**tf**.**zeros**([**10**]))**

y **=** tf**.**nn**.**softmax**(**tf**.**matmul**(**x**,** W**)** **+** b**)**

init\_op **=** tf**.**initialize\_all\_variables**()**

saver **=** tf**.**train**.**Saver**()**

with tf**.**Session**()** as sess**:**

sess**.**run**(**init\_op**)**

saver**.**restore**(**sess**,** "model.ckpt"**)**

prediction**=**tf**.**argmax**(**y**,**1**)**

**return** prediction**.**eval**(**feed\_dict**={**x**:** **[**imvalue**]},** session**=**sess**)**

#### 5.3.2 卷积神经网络的识别函数

def predictint**(**imvalue**):**

# Define the model (same as when creating the model file)

x **=** tf**.**placeholder**(**tf**.**float32**,** **[**None**,** 784**])**

W **=** tf**.**Variable**(**tf**.**zeros**([**784**,** 10**]))**

b **=** tf**.**Variable**(**tf**.**zeros**([**10**]))**

def weight\_variable**(**shape**):**

initial **=** tf**.**truncated\_normal**(**shape**,** stddev**=**0.1**)**

**return** tf**.**Variable**(**initial**)**

def bias\_variable**(**shape**):**

initial **=** tf**.**constant**(**0.1**,** shape**=**shape**)**

**return** tf**.**Variable**(**initial**)**

def conv2d**(**x**,** W**):**

**return** tf**.**nn**.**conv2d**(**x**,** W**,** strides**=[**1**,** 1**,** 1**,** 1**],** padding**=**'SAME'**)**

def max\_pool\_2x2**(**x**):**

**return** tf**.**nn**.**max\_pool**(**x**,** ksize**=[**1**,** 2**,** 2**,** 1**],** strides**=[**1**,** 2**,** 2**,** 1**],** padding**=**'SAME'**)**

W\_conv1 **=** weight\_variable**([**5**,** 5**,** 1**,** 32**])**

b\_conv1 **=** bias\_variable**([**32**])**

x\_image **=** tf**.**reshape**(**x**,** **[-**1**,**28**,**28**,**1**])**

h\_conv1 **=** tf**.**nn**.**relu**(**conv2d**(**x\_image**,** W\_conv1**)** **+** b\_conv1**)**

h\_pool1 **=** max\_pool\_2x2**(**h\_conv1**)**

W\_conv2 **=** weight\_variable**([**5**,** 5**,** 32**,** 64**])**

b\_conv2 **=** bias\_variable**([**64**])**

h\_conv2 **=** tf**.**nn**.**relu**(**conv2d**(**h\_pool1**,** W\_conv2**)** **+** b\_conv2**)**

h\_pool2 **=** max\_pool\_2x2**(**h\_conv2**)**

W\_fc1 **=** weight\_variable**([**7 **\*** 7 **\*** 64**,** 1024**])**

b\_fc1 **=** bias\_variable**([**1024**])**

h\_pool2\_flat **=** tf**.**reshape**(**h\_pool2**,** **[-**1**,** 7**\***7**\***64**])**

h\_fc1 **=** tf**.**nn**.**relu**(**tf**.**matmul**(**h\_pool2\_flat**,** W\_fc1**)** **+** b\_fc1**)**

keep\_prob **=** tf**.**placeholder**(**tf**.**float32**)**

h\_fc1\_drop **=** tf**.**nn**.**dropout**(**h\_fc1**,** keep\_prob**)**

W\_fc2 **=** weight\_variable**([**1024**,** 10**])**

b\_fc2 **=** bias\_variable**([**10**])**

y\_conv**=**tf**.**nn**.**softmax**(**tf**.**matmul**(**h\_fc1\_drop**,** W\_fc2**)** **+** b\_fc2**)**

init\_op **=** tf**.**initialize\_all\_variables**()**

saver **=** tf**.**train**.**Saver**()**

with tf**.**Session**()** as sess**:**

sess**.**run**(**init\_op**)**

saver**.**restore**(**sess**,** "model2.ckpt"**)**

#print ("Model restored.")

prediction**=**tf**.**argmax**(**y\_conv**,**1**)**

**return** prediction**.**eval**(**feed\_dict**={**x**:** **[**imvalue**],**keep\_prob**:** 1.0**},** session**=**sess**)**