摘要

# 本周周报的主要内容是由于莫烦的python视频对于基本理论性的内容并没有详细地进行阐述，也为了快速入门这周我学习TensorFlow官方文档中文版、翻译概括了部分RoboMozart关于LSTM产生音乐的论文。

# MNIST机器学习入门

http://wiki.jikexueyuan.com/project/tensorflow-zh/tutorials/mnist\_beginners.html

# MNIST数据集是一个手写数字图像的数据集，包含了60000个训练样本和10000个[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/wspba/article/details/_blank)样本，每个样本的维度是784（28\*28），样本的值代表图像的灰度值，值的范围为0~1，每个样本对应一个标签，标签是一个长度为10的向量，向量中对应数字真值的位置为1，其余为0。

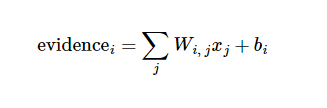
import tensorflow.examples.tutorials.mnist.input\_data

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

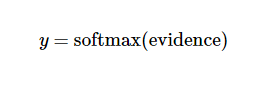
## Softmax回归介绍

softmax回归（softmax regression）分两步：第一步

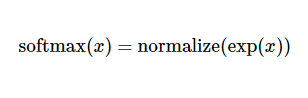
为了得到一张给定图片属于某个特定数字类的证据（evidence），我们对图片像素值进行加权求和。如果这个像素具有很强的证据说明这张图片不属于该类，那么相应的权值为负数，相反如果这个像素拥有有利的证据支持这张图片属于这个类，那么权值是正数。



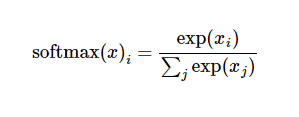
softmax函数可以把这些证据转换成概率****y****：



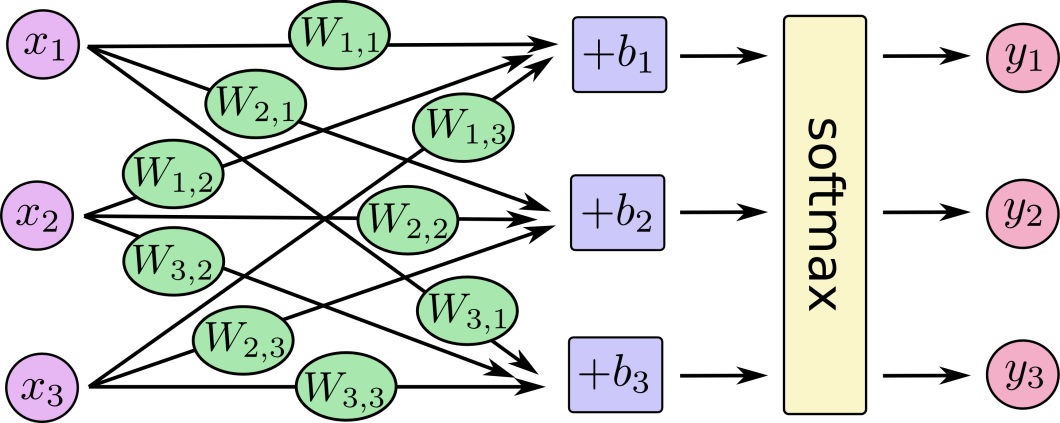
这里的softmax可以看成是一个激励（activation）函数或者链接（link）函数，把我们定义的线性函数的输出转换成我们想要的格式



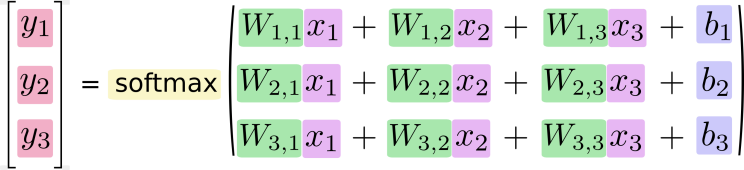
展开右边的子式：



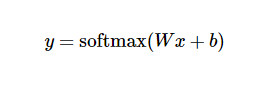
把输入值当成幂指数求值，再正则化这些结果值。这个幂运算表示，更大的证据对应更大的假设模型（hypothesis）里面的乘数权重值。反之，拥有更少的证据意味着在假设模型里面拥有更小的乘数系数。假设模型里的权值不可以是0值或者负值。Softmax然后会正则化这些权重值，使它们的总和等于1，以此构造一个有效的概率分布。



转换为数学表达式如下：



最终能写成如下：



## 实现回归模型

import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 784])#构建占位符，代表输入的图像，None表示样本的数量可以是任意的

W = tf.Variable(tf.zeros([784,10])) #构建一个变量，代表训练目标W，初始化为0

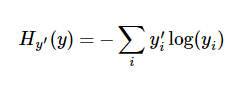
b = tf.Variable(tf.zeros([10])) #构建一个变量，代表训练目标b，初始化为0

y = tf.nn.softmax(tf.matmul(x,W) + b) #构建了一个softmax的模型：y = softmax(Wx + b)，y指样本标签的预测值

## 训练模型

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）

利用交叉熵作为成本函数



y\_ = tf.placeholder("float", [None,10])  #构建占位符，代表样本标签的真实值

计算交叉熵:

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y)) #交叉熵代价函数

train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)#使用梯度下降法（0.01的学习率）来最小化这个交叉熵代价函数

init = tf.initialize\_all\_variables()

sess = tf.Session() #构建会话

sess.run(init)#初始化所有变量

for i in range(1000):

batch\_xs, batch\_ys = mnist.train.next\_batch(100) #使用minibatch的训练数据，一个batch的大小为100

sess.run(train\_step, feed\_dict={x: batch\_xs, y\_: batch\_ys}） #用训练数据替代占位符来执行训练

correct\_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y\_,1))#tf.argmax()返回的是某一维度上其数据最大所在的索引值，在这里即代表预测值和真值

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))#用平均值来统计测试准确率

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

# 深入MNIST

TensorFlow是一个非常强大的用来做大规模数值计算的库。其所擅长的任务之一就是实现以及训练深度神经网络。

在本教程中，我们将学到构建一个TensorFlow模型的基本步骤，并将通过这些步骤为MNIST构建一个深度卷积神经网络。

### 加载MNIST数据

import input\_data

mnist = input\_data.read\_data\_sets('MNIST\_data', one\_hot=True)

### 运行TensorFlow的InteractiveSession

Tensorflow依赖于一个高效的C++后端来进行计算。与后端的这个连接叫做session。一般而言，使用TensorFlow程序的流程是先创建一个图，然后在session中启动它。]

所使用的是ReLU神经元

<http://www.cnblogs.com/neopenx/p/4453161.html>

import tensorflow as tf

sess = tf.InteractiveSession()

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)

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。

卷积层的作用是提取局部特征，池化的作用是降维

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)

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

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

/\*把池化层输出的张量reshape成一些向量，乘上权重矩阵，加上偏置，然后对其使用ReLU。\*/

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)

为了减少过拟合，我们在输出层之前加入dropout。

TensorFlow的tf.nn.dropout操作除了可以屏蔽神经元的输出外，还会自动处理神经元输出值的scale。所以用dropout的时候可以不用考虑scale。

keep\_prob = tf.placeholder("float")

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)

### 训练和评估模型

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

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

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

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

sess.run(tf.initialize\_all\_variables())

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})

关于RoboMozart关于LSTM产生音乐这篇论文。对文章部分进行了翻译，也大致了解了总体上如何用LSTM来产生音乐。

1. 使用MIDI来存储音乐
2. 去除MIDI文件中冗余的数据
3. 使用MIDI来训练神经网络

论文中对比了其他神经网络架构用来产生音乐的研究。

从进化算法、LSTM、RNN、LSTM循环网络

最终得出选择LSTM神经网络，选择单一乐队来训练神经网络，所能达到后续音乐的预测的准确度较高。

使用MIDI文件的原因：

1. MIDI文件是一种描述性的“[音乐语言](https://baike.baidu.com/item/%E9%9F%B3%E4%B9%90%E8%AF%AD%E8%A8%80" \t "https://baike.baidu.com/item/MIDI/_blank)”，它将所要演奏的乐曲信息用[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/MIDI/_blank)进行描述。譬如在某一时刻，使用什么乐器，以什么音符开始，以什么音调结束，加以什么伴奏等等，MIDI文件本身并不包含[波形](https://baike.baidu.com/item/%E6%B3%A2%E5%BD%A2" \t "https://baike.baidu.com/item/MIDI/_blank)数据，所以MIDI文件非常小巧。（来自百度）
2. 文件相对于单纯的波形数据来说小很多。

由于对于Encoding的代码还没有完全理解所以暂时不在此次周报中描述。

下一周将要完成该篇论文代码的理解和tensorflow更加深入的学习。