

**华东理工大学**

模式识别课程报告

—— BP神经网络、卷积神经网络原理介绍

与图像识别应用

**题目** CIFAR-10十分类

**院系** 信息科学与工程学院

**专业**  控制科学与工程

**组员**  王志强

**指导老师**  赵海涛

目录

[一、 选题介绍 1](#_Toc4409)

[1.1 原kaggle选题 1](#_Toc13916)

[1.2 CIFAR-10数据集介绍 2](#_Toc20546)

[二、 实验目的 2](#_Toc1870)

[三、 原理介绍 3](#_Toc12766)

[3.1 BP神经网络 3](#_Toc26946)

[3.1.1 神经元模型 3](#_Toc27464)

[3.1.2 误差反向传播 3](#_Toc17489)

[3.1.3 更新权重 4](#_Toc21722)

[3.1.4 BP神经网络于图像识别 4](#_Toc25987)

[3.2 卷积神经网络 4](#_Toc27946)

[3.2.1 卷积核与卷积 5](#_Toc26628)

[3.2.2 卷积层 5](#_Toc17909)

[3.2.3 池化层 6](#_Toc29908)

[3.2.4 激活层 6](#_Toc12688)

[3.2.5 全连接层 7](#_Toc29952)

[3.2.6 交叉熵损失 7](#_Toc31927)

[3.2.7 参数更新 7](#_Toc29356)

[3.2.8 Tensorflow 9](#_Toc8174)

[四、 实验结论与分析 9](#_Toc1914)

[4.1 BP神经网络结果与分析 9](#_Toc27856)

[4.2 卷积神经网络结果与分析 10](#_Toc27406)

[五、 总结 12](#_Toc7942)

[六、 参考文献 13](#_Toc2746)

[Python代码附录 14](#_Toc13751)

[1、 BP神经网络对kaggle选题 14](#_Toc7074)

[2、 BP神经网络对CIFAR-10 18](#_Toc25006)

[3、 卷积神经网络对CIFAR-10 20](#_Toc25295)

1. **选题介绍**

1.1 原kaggle选题

题目网址：<https://www.kaggle.com/c/image-classification2>

最初选定的题目是kaggle上的一道彩色图像分类题目，即有三类图片：汽车、摩托车和自行车；要求参与者正确识别它们，方法不限。数据集有：训练集包含231张自行车图片、518张汽车图片和218张摩托车图片；测试集包含了914张三类混合的图片。数据实例如图1所示。



图1 训练集选图

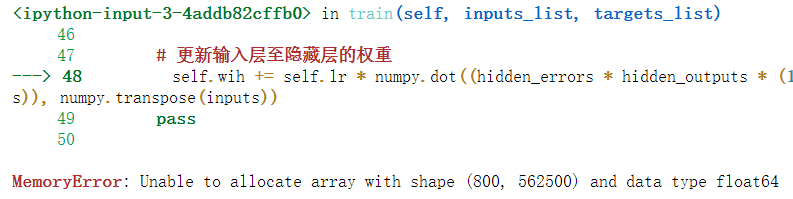


图2 jupyter notebook报错

然后在接下来的实验中遇到了内存不足的问题。如图2 所示，编译器提示“储存错误：无法分配800×562500的数组来存储浮点型数字数据”。题目所给的图片规模为：500\*375\*3，若用神经网络的方法则会产生数量庞大的变量。在我将电脑虚拟内存设置成最大后仍有这个问题。

最终我选择将数据集换成经典的、处理数据规模较小CIFAR-10来测试神经网络的性能。

1.2 CIFAR-10数据集介绍

CIFAR-10数据集由10个类的60000个32x32三通道彩色图像组成，每个类有6000个图像。总共有50000个训练图像和10000个测试图像。十个类别为：飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船和卡车。

数据集分为五个训练批次和一个测试批次，每个批次有10000个图像。测试批次包含来自每个类别的恰好1000个随机选择的图像。训练批次以随机顺序包含剩余图像，但一些训练批次可能包含来自一个类别的图像比另一个更多。总体来说，所有训练批组成的训练集，每一类都有5000张图。这些类完全相互排斥。汽车和卡车之间没有重叠。“汽车”包括轿车，SUV，这类东西。“卡车”只包括大卡车。都不包括皮卡车。

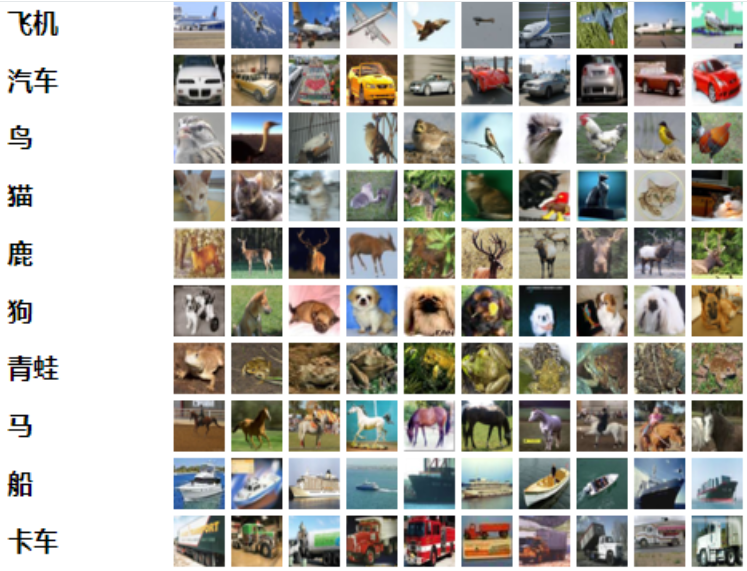


图3 CIFAR-10图像随机举例

1. **实验目的**

运用BP神经网络和卷积神经网络于CIFAR-10数据集进行图像识别的训练与测试，分析两种方法的实验结果。

实验工具：jupyter notebook ，python。

1. **原理介绍**

人工神经网络（Artificial Neutral Network，ANN）是由大量简单的基本元件——神经元相互连接，通过模拟人的大脑神经处理信息的方式，进行信息并行处理和非线性转换的复杂网络系统。

3.1 BP神经网络

3.1.1 神经元模型

人造神经元模型如图 4 所示。该神经元的输入输出关系为：



其中  为神经元的输入， 为各神经元之间的权重，b为神经元的阈值，y为神经元的输出，为神经元的激活函数，常用的转化函数如图 5 所示。

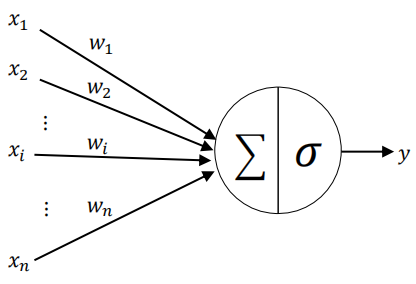


图4 人造神经元

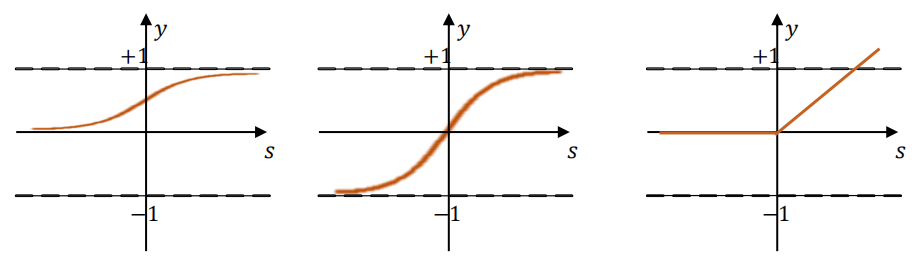


图5 sigmoid、tanh、ReLU函数

3.1.2 误差反向传播

三层神经网络实例如图6。BP神经网络误差的反向传播如图7所示。在BP神经网络中可以选择不等分误差传播，即为较大链接权重的连接分配更多的误差。因为这些链接对造成误差的贡献较大。如图7，第一层1号节点至下层1号节点的权重为3.0，占总权重的3/4，所以传播给它的误差是3/4的输出误差。

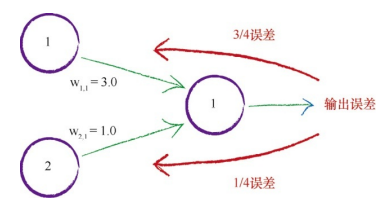
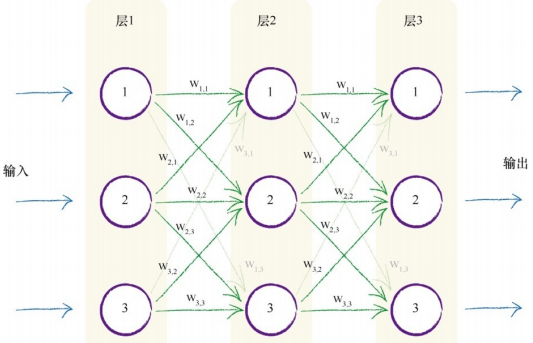


图6、7 三层神经网络与误差反向传播

3.1.3 更新权重

可以运用梯度下降法更新神经网络中的权重参数。



式中，称为学习率。这个因子可以调节权重变化的强度，确保不会超调。

3.1.4 BP神经网络于图像识别

在BP神经网络的输入层中，神经元个数是图像的每个像素。例如一个375\*500\*3的彩色图像，把它依次拉成列向量作为输入，也就是共有375\*500\*3=562500个神经元。具体用python代码实现如下。

|  |
| --- |
| # 将三维数据矩阵拉成列向量 |
| for i in range(len(d\_bicycle)):  t1=d\_bicycle[i].reshape(375\*500\*3,)  t1=t1.reshape(-1,1)  if i==0:  da\_bicycle=t1  if i!=0:  da\_bicycle=np.append(da\_bicycle,t1,axis=1) |

3.2 卷积神经网络

卷积神经网络（Convolutional Neural Network，CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。 它包括卷积层(convolutional layer)、池化层(pooling layer)和全连接层(fully connected layer)。

在图像识别中，图像的像素数量是网络输入层神经元的数量；卷积后得到的像素个数是卷积层中的神经元个数；池化后得的数据个数是池化层的神经元个数；参数更新是指对卷积核内权重以及全连接层中的权重数据的更新。

3.2.1 卷积核与卷积

卷积核又称滤波器（filter）。卷积作用以图8为例：输入数据是一个8×8的矩阵，卷积核是一个大小为3×3的矩阵，将卷积核在输入矩阵上从左向右、至上而下的平移，每次平移后，卷积核与新的部分输入数据重合对应。卷积作用就是将输入数据对应的9个数据加权求和得到一个数据。如图8，设卷积核滑动步长为1，则卷积后得到一个6×6的输出矩阵。

卷积核中的数据决定了卷积核的作用。如图8的卷积核，它就可以用来检测图像边缘。

卷积核中的数字变量是卷积神经网络中需要训练的权重参数之一。

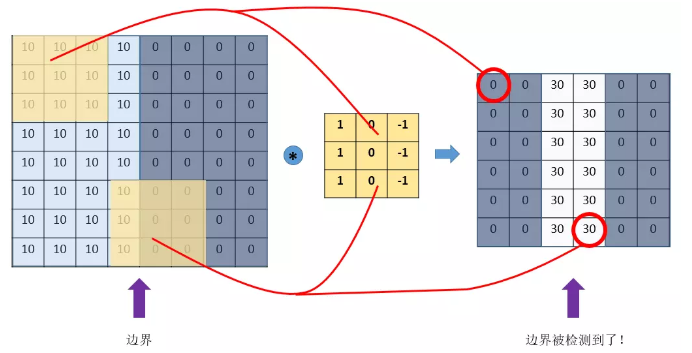


图8 卷积作用

3.2.2 卷积层

卷积层就是用卷积核对输入图像数据进行卷积的阶段。以CIFAR-10数据集为例：每张图是32\*32\*3的三维数据，即长度和宽度为32个像素，颜色通道为RGB三通道；也就是说一张图的输入数据是三个32×32的数字矩阵，相应地，卷积核也是三维矩阵，例如三个3×3的矩阵，假设滑动步长为1，那么卷积后就会得到三个30×30的输出矩阵。

一个卷积层常用多个卷积核。如图9是一个三通道图片的卷积过程，有四个卷积核，大小为3×3×3，8×8×3的输入数据经过卷积后得到6×6×4（也可以说是6×6×3×4）的输出。

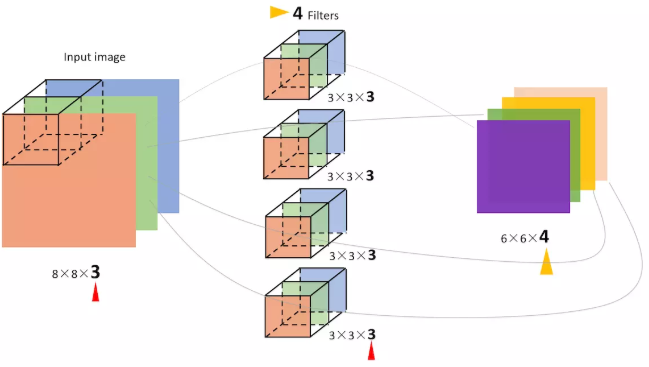


图9 多通道图片的卷积

3.2.3 池化层

池化作用是为了提取一定区域的主要特征，并减少参数数量，防止模型过拟合。常用的有Maxpooling，采用一个2×2的窗口，并设置滑动步长为2，如图10所示。即取窗口中最大的数据作为输出数据，是一种下采样方式。

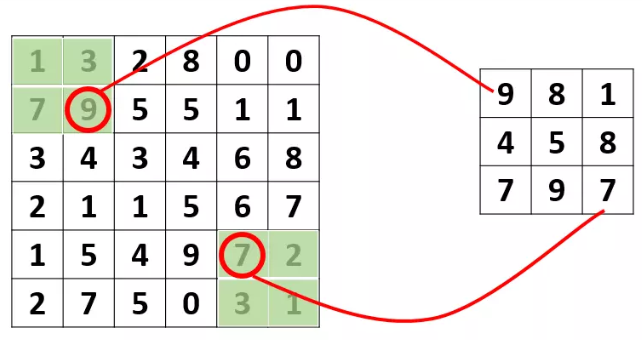


图10 MAX池化

3.2.4 激活层

卷积神经网络中的激活概念和BP神经网络中的是一致的。本次实验使用ReLU函数作为激活函数。

3.2.5 全连接层

全连接层的形式和BP神经网络中的隐藏层与输出层的形式一致。因为这一层是每一个单元都和前一层的每一个单元相连接，所以称之为“全连接”。具体实现中就是将数据矩阵依次拉成列向量后连接成为一个超长的列向量，每个数据都作为一个神经元，其输出就是数据数值，其余和BP神经网络一致。

3.2.6 交叉熵损失

举个例子。P用表示样本的真实分布，比如[1，0，0]表示当前样本属于第一类。Q用来表示神经网络预测的分布，比如[0.7，0.2，0.1]。直观的理解就是如果用P来描述样本就十分完美。若用Q来描述样本，信息量不足，需要额外的一些“信息增量”才能达到如P一样的描述。可以通过训练Q，使其像P一样描述样本。用KL散度来描述、的联系：



式中，，表示类为第i类的概率，对数据规范化后也就是的数值，形式一致。等式的前一部分是的熵，后一部分就是交叉损失熵：



3.2.7 参数更新

卷积神经网络的参数更新也可以使用梯度下降法，且在理论上的推导是和BP神经网络相一致的。

**全连接层梯度反向传播**：

全连接层之间传播可与BP神经网络一样，使用不等分误差传播。

假设，是输入，是权重，是偏置，是输出分值矩阵。若已知，是损失函数，则：，，中每个元素是每列之和。

**激活层梯度反向传播**：

假设采用ReLU作为激活函数。公式描述为：，其中激活层输入和输出。该公式是逐元计算的。已知，求。因为当输入小于0是，梯度为0，当输入大于0时，梯度为1。所以的每个元素只需与0比较，若大于0，则输出梯度等于输入梯度，否则为0。

**卷积层梯度的反向传播**：

先简述利用矩阵乘法实现卷积层正向计算的过程：

1. 将输入特征图变成大矩阵。
2. 进行矩阵相乘和非线性激活后得到输出数据。
3. 将输出数据变换为输出特征图。

卷积层梯度传播就是已知输出特征图的梯度，求输入特征图的梯度及卷积核的梯度，其过程如下。

1. 把输出特征图的梯度变换为矩阵形式（正向计算第三步的逆过程）。
2. 将全连接层和激活层的梯度进行反向传播。
3. 把第（2）步得到的矩阵梯度变换为特征图形状的梯度，即得到输入特征图的梯度。

**卷积层梯度的反向传播**：

最大池化层是求的偏导数，其中a,b,c,d是池化窗口中的数据。该函数的偏导数：a,b,c,d中的最大值的梯度为1，其余为0。 推到如下：

，

当a为最大值时，则上式变为：；当a不为最大值，不妨设b为最大值，则上式变为。

所以，池化层梯度反向传播时，每个局部窗口中，若该元素是最大值，则该位置处的输出梯度等于输入梯度，否则为0。公式原理如下：





3.2.8 Tensorflow

TensorFlow是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库DistBelief。Tensorflow拥有多层级结构，可部署于各类服务器、PC终端和网页并支持GPU和TPU高性能数值计算，被广泛应用于谷歌内部的产品开发和各领域的科学研究。TensorFlow由谷歌人工智能团队谷歌大脑（Google Brain）开发和维护，拥有包括TensorFlow Hub、TensorFlow Lite、TensorFlow Research Cloud在内的多个项目以及各类应用程序接口（Application Programming Interface, API）。

本次实验卷积神经网络的实现使用了库tensorflow。

1. **实验结论与分析**

4.1 BP神经网络结果与分析

本次实验采用三层反馈神经网络。实验结果如表1所示。可见三层反馈神经网络对于十分类识别还是太勉强了，最高的准确率也只有近 25% 。但是BPNN是有效果的，大于10% 就是比蒙好。而且有意思的是，BP网络随着隐藏层神经元个数的增加，性能是好后坏的。这应该是对应着的过多的神经元会造成对训练集过拟合而对测试集表现不佳的情况。在这种超出网络能力的情况下学习率是越小越好的。

表1 BP神经网络实验数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入层神经元个数 | 隐藏层神经元个数 | 输出层神经元个数 | 学习率 | 准确率 |
| 3072 | 175 | 10 | 0.6 | 0.1054 |
| 3072 | 10 | 0.3 | 0.1472 |
| 3072 | 10 | 0.2 | 0.1610 |
| 3072 | 10 | 0.1 | 0.2404 |
| 3072 | 300 | 10 | 0.1 | 0.2468 |
| 3072 | 400 | 10 | 0.2262 |
| 3072 | 600 | 10 | 0.2358 |
| 3072 | 1000 | 10 | 0.2220 |

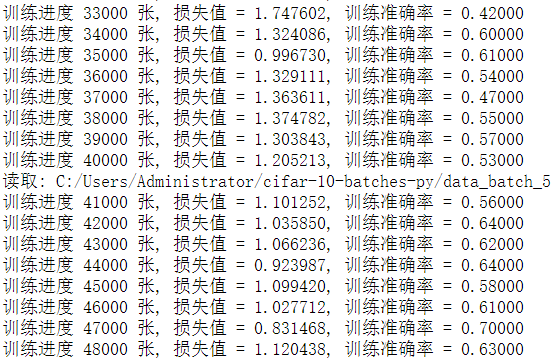
4.2 卷积神经网络结果与分析

本次实验所使用的卷积神经网络规模及策略：分批次每次训练100张图片；网络结构：卷积层1+卷积层2+卷积层3+全连接层1+全连接层2；各个卷积层都含有池化层和激活层，采用最大值池化和ReLU激活函数；卷积层1用32个3×3×3卷积核，卷积层2用64个3×3×32卷积核，卷积层3用128个3×3×64卷积核；损失函数用交叉熵损失；参数优化用tensorflow的AdamOptimizer优化器。

实验结果如图11所示。对于CIFAR-10数据集，该卷积神经网络的识别准确率为64.44%，相比BP神经网络有了巨大的改善。从每个批次（10000张）训练过程来看，训练准确率提升至50%还是比较快的，第一个批次中就有出现。然而至60%以上的准确率就比较缓慢。也就是说卷积神经网络的训练效果(正确率)是斜率减小的上升曲线，很有可能稳定收敛于一个值，那么这个收敛值距离100%的距离就确定了该神经网络在理论上对该问题是否能到达完美的表现。

增加卷积核的数目可以提高网络的准确率，总结如表2所示。对于更大规模的网络我的电脑就跑不动了。

卷积神经网络在图像识别上有巨大的优势主要是归功于卷积有能增大神经元视野等功能，和卷积神经网络的性质契合了图像的多层次结构、特征局部性、平移不变性三个特性。除此之外，还有一个有趣的辅助功能是遗忘率，模仿人类记忆的遗忘，一次迭代中是全连接层中一定比例的神经元失去参数更新的能力，能防止网络对训练集过拟合，从而具备更好的广泛性。



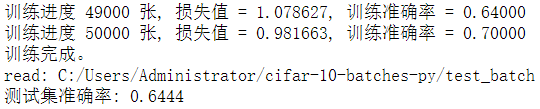


图11 卷积神经网络运行结果

表2 不同卷积核数目对比

|  |  |  |  |
| --- | --- | --- | --- |
| 准确率 | 卷积层1  卷积核数目 | 卷积层2  卷积核数目 | 卷积层3  卷积核数目 |
| 0.5579 | 8 | 16 | 32 |
| 0.6160 | 16 | 32 | 64 |
| 0.6444 | 32 | 64 | 128 |
| 0.6547 | 64 | 128 | 256 |

1. **总结**

本次实验介绍了CIFAR-10数据集、BP神经网络和卷积神经网络的原理；运用网络对CIFAR-10数据集训练、测试；分析了BP神经网络和卷积神经网络在CIFAR-10数据集上的表现效果。根据实验结果，卷积神经网络在图像识别上的表现比BP神经网络优越得多。

卷积神经网络虽然在数学、神经学等理论根据上肯定难得不行，但是在形式上并不难理解。若想编程实现，有BP神经网络的编程基础和对卷积神经网络形式有清楚的理解，应该就可以实现。但是由于大量繁琐的细节，编程耗时会很长。Tensorflow是一款很好的学习机器学习（例如神经网络）的工具，使用它可以极大地简化卷积神经网络的编程实现。

1. **参考文献**

1、《卷积神经网络的Python实现》 著者：单建华著；出版日期：2019 ；出版社：人民邮电出版社。

2、《TensorFlow与卷积神经网络从算法入门到项目实战》 著者：华超编著；出版日期：2019；出版社：电子工业出版社。

**Python代码附录**

1. BP神经网络对kaggle选题

|  |
| --- |
| # 读取图像所需  from PIL import Image  import numpy as np  import matplotlib.pyplot as plt  import os  import numpy  import scipy.special  import csv  import matplotlib.pyplot  %matplotlib inline  # BP神经网络类  class neuralNetwork:  def \_\_init\_\_(self, inputnodes, hiddennodes, outputnodes, learningrate):  self.inodes = inputnodes  self.hnodes = hiddennodes  self.onodes = outputnodes  # 权重矩阵，wih和who  #从节点i至下一层节点j的权重为w\_i\_j  self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))  self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))  # 学习率  self.lr = learningrate  # sigmoid作为激活函数  self.activation\_function = lambda x: scipy.special.expit(x)  pass  # 神经网络的训练  def train(self, inputs\_list, targets\_list):  # 输入行向量特征，处理为列向量  inputs = numpy.array(inputs\_list, ndmin=2).T  targets = numpy.array(targets\_list, ndmin=2).T  # 计算输入隐藏层的信号  hidden\_inputs = numpy.dot(self.wih, inputs)  # 计算隐藏层输出信号  hidden\_outputs = self.activation\_function(hidden\_inputs)  # 计算输出层输入信号  final\_inputs = numpy.dot(self.who, hidden\_outputs)  # 计算最终输出信号  final\_outputs = self.activation\_function(final\_inputs)  # 输出层误差为 (target - actual)  output\_errors = targets - final\_outputs  # 计算隐藏层输出误差  hidden\_errors = numpy.dot(self.who.T, output\_errors)  # 更新隐藏层至输出层的权重  self.who += self.lr \* numpy.dot((output\_errors \* final\_outputs \* (1.0 - final\_outputs)), numpy.transpose(hidden\_outputs))  # 更新输入层至隐藏层的权重  self.wih += self.lr \* numpy.dot((hidden\_errors \* hidden\_outputs \* (1.0 - hidden\_outputs)), numpy.transpose(inputs))  pass  # 运行神经网络  def query(self, inputs\_list):  inputs = numpy.array(inputs\_list, ndmin=2).T  hidden\_inputs = numpy.dot(self.wih, inputs)  hidden\_outputs = self.activation\_function(hidden\_inputs)  final\_inputs = numpy.dot(self.who, hidden\_outputs)  final\_outputs = self.activation\_function(final\_inputs)  return final\_outputs  # main  # 读图像文件函数  def load\_Img(imgDir,imgFoldName):  imgs = os.listdir(imgDir+imgFoldName)  imgNum = len(imgs) #文件夹中图片数量  img0 = Image.open(imgDir+imgFoldName+"/"+imgs[0])  arr0 = np.array(img0)  [a1,a2,a3]=arr0.shape  data = np.empty((imgNum,a1,a2,a3))  for i in range (imgNum):  img = Image.open(imgDir+imgFoldName+"/"+imgs[i])  arr = np.array(img)  data[i,:,:,:] = arr # 依次读取图片转换成像素图  return data  # 读图像  craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"  foldName = "bicycle"  d\_bicycle=load\_Img(craterDir,foldName)  craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"  foldName = "car"  d\_car=load\_Img(craterDir,foldName)  craterDir = "C:/Users/Administrator/Kaggle/选取的训练集/"  foldName = "motorbike"  d\_motorbike=load\_Img(craterDir,foldName)  # 将三维数据矩阵拉成列向量  for i in range(len(d\_bicycle)):  t1=d\_bicycle[i].reshape(375\*500\*3,)  t1=t1.reshape(-1,1)  if i==0:  da\_bicycle=t1  if i!=0:  da\_bicycle=np.append(da\_bicycle,t1,axis=1)  # 将三维数据矩阵拉成列向量  for i in range(len(d\_motorbike)):  t1=d\_motorbike[i].reshape(375\*500\*3,)  t1=t1.reshape(-1,1)  if i==0:  da\_motorbike=t1  if i!=0:  da\_motorbike=np.append(da\_motorbike,t1,axis=1)  da\_car=da\_car.T;da\_motorbike=da\_motorbike.T  # number of input, hidden and output nodes  input\_nodes = 562500  hidden\_nodes = 800  output\_nodes = 3  # learning rate  learning\_rate = 0.3  # create instance of neural network  n = neuralNetwork(input\_nodes,hidden\_nodes,output\_nodes, learning\_rate) #BP神经网络框架  #读入bicycle训练  epochs = 1  for e in range(epochs):  for record in da\_bicycle:  all\_values=record  inputs = all\_values / 255.0 \* 0.99 + 0.01 #所有数据归一化  targets = numpy.zeros(output\_nodes) + 0.01  targets[0] = 0.99  n.train(inputs, targets)  pass  Pass  #读入car训练  epochs = 1  for e in range(epochs):  for record in da\_car:  all\_values=record  inputs = all\_values / 255.0 \* 0.99 + 0.01 #所有数据归一化  targets = numpy.zeros(output\_nodes) + 0.01  targets[1] = 0.99  n.train(inputs, targets)  pass  Pass  #读入motorbike训练  epochs = 1  for e in range(epochs):  for record in da\_motorbike:  all\_values=record  inputs = all\_values / 255.0 \* 0.99 + 0.01 #所有数据归一化  targets = numpy.zeros(output\_nodes) + 0.01  targets[2] = 0.99  n.train(inputs, targets)  pass  Pass  # load the mnist test data CSV file into a list  # 转载测试集  craterDir = "C:/Users/Administrator/Kaggle/"  foldName = "选取的测试集"  d\_test=load\_Img(craterDir,foldName)  for i in range(len(d\_test)):  t1=d\_test[i].reshape(375\*500\*3,)  t1=t1.reshape(-1,1)  if i==0:  da\_test=t1  if i!=0:  da\_test=np.append(da\_test,t1,axis=1)  da\_test=da\_test.T  da\_test.shape  # 测试神经网络  # 记录判别状态用数 组  scorecard = []  for record in da\_test:  all\_values = record  correct\_label = que\_ding\_de\_zhi  inputs = all\_values / 255.0 \* 0.99 + 0.01  # 询问输出层  outputs = n.query(inputs)  # 取最大值为神经网络判断的类别  label = numpy.argmax(outputs)  # 判断正确增加1，否则增加0  if (label == correct\_label):  scorecard.append(1)  else:  scorecard.append(0)  pass  Pass  # 计算正确率  scorecard\_array = numpy.asarray(scorecard)  print ("正确率 = ", scorecard\_array.sum() / scorecard\_array.size) |

1. BP神经网络对CIFAR-10

|  |
| --- |
| from PIL import Image  import numpy as np  import os  from \_\_future\_\_ import print\_function  from six.moves import cPickle as pickle  from scipy.misc import imread  import platform  import numpy  import scipy.special  import csv  import matplotlib.pyplot  %matplotlib inline  import cifar\_10\_read # 用来装载cifar-10  [train\_data,train\_label,test\_data,test\_label]=cifar\_10\_read.load\_CIFAR10("C:/Users/Administrator/cifar-10-batches-py/")  # BP神经网络类  class neuralNetwork:  def \_\_init\_\_(self, inputnodes, hiddennodes, outputnodes, learningrate):  self.inodes = inputnodes  self.hnodes = hiddennodes  self.onodes = outputnodes  self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))  self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))  self.lr = learningrate  self.activation\_function = lambda x: scipy.special.expit(x)  pass  def train(self, inputs\_list, targets\_list)  inputs = numpy.array(inputs\_list, ndmin=2).T  targets = numpy.array(targets\_list, ndmin=2).T  hidden\_inputs = numpy.dot(self.wih, inputs)  hidden\_outputs = self.activation\_function(hidden\_inputs)  final\_inputs = numpy.dot(self.who, hidden\_outputs)  final\_outputs = self.activation\_function(final\_inputs)  output\_errors = targets - final\_outputs  hidden\_errors = numpy.dot(self.who.T, output\_errors)  self.who += self.lr \* numpy.dot((output\_errors \* final\_outputs \* (1.0 - final\_outputs)), numpy.transpose(hidden\_outputs))  self.wih += self.lr \* numpy.dot((hidden\_errors \* hidden\_outputs \* (1.0 - hidden\_outputs)), numpy.transpose(inputs))  pass  def query(self, inputs\_list):  inputs = numpy.array(inputs\_list, ndmin=2).T  hidden\_inputs = numpy.dot(self.wih, inputs)  hidden\_outputs = self.activation\_function(hidden\_inputs)  final\_inputs = numpy.dot(self.who, hidden\_outputs)  final\_outputs = self.activation\_function(final\_inputs)  return final\_outputs  # number of input, hidden and output nodes  input\_nodes = 3072  hidden\_nodes = 1000  output\_nodes = 10  # learning rate  learning\_rate = 0.1  # create instance of neural network  n = neuralNetwork(input\_nodes,hidden\_nodes,output\_nodes, learning\_rate) #BP神经网络框架  da\_train = numpy.loadtxt(open("C:/Users/Administrator/作业/featvector.csv","rb"), delimiter=",", skiprows=0)  da\_test = numpy.loadtxt(open("C:/Users/Administrator/作业/featvector\_test.csv","rb"), delimiter=",", skiprows=0)  jishu=0  for record in da\_train:  all\_values=record  inputs = all\_values / 255.0 \* 0.99 + 0.01 #所有数据归一化  targets = numpy.zeros(output\_nodes) + 0.01  targets[train\_label[jishu]] = 0.99  n.train(inputs, targets)  jishu=jishu+1  Pass  # 测试神经网络  # 记录判别状态用数组  jishu=0  scorecard = []  for record in da\_test:  all\_values = record  correct\_label = test\_label[jishu]  inputs = all\_values / 255.0 \* 0.99 + 0.01  # 询问输出层  outputs = n.query(inputs)  # 取最大值为神经网络判断的类别  label = numpy.argmax(outputs)  # 判断正确增加1，否则增加0  if (label == correct\_label):  scorecard.append(1)  else:  scorecard.append(0)  pass  pass  jishu=jishu+1  # 计算正确率  scorecard\_array = numpy.asarray(scorecard)  print ("正确率 = ", scorecard\_array.sum() / scorecard\_array.size) |

1. 卷积神经网络对CIFAR-10

|  |
| --- |
| import tensorflow.compat.v1 as tf # 需要用tensorflow  tf.disable\_v2\_behavior()  import cifar\_reader # 读取cifar-10的函数  batch\_size = 100 # 每次处理100个图片，分批处理  s = 0  show\_s = 10  train\_iter = 50000 # 共50000张训练图片  # placeholder()函数是在神经网络构建graph的时候在模型中的占位，此时并没有把要输入的数据传入模型，  # 它只会分配必要的内存。等建立session，在会话中，运行模型的时候通过feed\_dict()函数向占位符喂入数据。  input = tf.placeholder(dtype=tf.float32, shape=[None, 32, 32, 3]) # 给input\_x占个位 ，None个数不确定  y\_lab = tf.placeholder(dtype=tf.float32, shape=[None, 10]) # 输出分值  rember = tf.placeholder(tf.float32) # 遗忘率  train\_sign = tf.placeholder(tf.bool)  #### 卷积层1  ## tf.Variable()  ## 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。  # 这个函数产生正太分布，均值和标准差自己设定  W1 = tf.Variable(tf.truncated\_normal([3, 3, 3, 32], dtype=tf.float32, stddev=1e-2))  # 随机初始化权重，使用64个卷积核  '''  tf.nn.conv2d(input, filter, strides, padding, use\_cudnn\_on\_gpu=None, name=None)  除去name参数用以指定该操作的name，与方法有关的一共五个参数：  input：指需要做卷积的输入图像，它要求是一个Tensor，具有[batch, in\_height, in\_width, in\_channels]这样的shape，  具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，注意这是一个4维的Tensor，要求类型为float32和float64其中之一  filter：相当于CNN中的卷积核，它要求是一个Tensor，具有[filter\_height, filter\_width, in\_channels, out\_channels]这样的shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数input相同，有一个地方需要注意，第三维in\_channels，就是参数input的第四维  strides：卷积时在图像每一维的步长，这是一个一维的向量，长度4  padding：string类型的量，只能是"SAME","VALID"其中之一  结果返回一个Tensor，这个输出，就是我们常说的feature map，shape仍然是[batch, height, width, channels]这种形式。  '''  conv1 = tf.nn.conv2d(input, W1, strides=(1, 1, 1, 1), padding="VALID")  # padding="VALID"，窗口移动，数据不足时直接舍弃  # BN 批量规范化  bn1 = tf.layers.batch\_normalization(conv1, training=train\_sign)  # 这个函数tf.nn.relu()的作用是计算激活函数 relu，即 max(features, 0):将大于0的保持不变，小于0的数置为0。  relu1 = tf.nn.relu(bn1)  # 池化层，选用最大法池化  pool1 = tf.nn.max\_pool(relu1, strides=[1, 2, 2, 1], padding="VALID", ksize=[1, 3, 3, 1])  '''tf.nn.max\_pool(value, ksize, strides, padding, name=None)  参数是四个，和卷积很类似：  第一个参数value：需要池化的输入，一般池化层接在卷积层后面，所以输入通常是feature map，依然是  [batch, height, width, channels]这样的shape  第二个参数ksize：池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]，因为我们不想在  batch和channels上做池化，所以这两个维度设为了1  第三个参数strides：和卷积类似，窗口在每一个维度上滑动的步长，一般也是[1, stride,stride, 1]  第四个参数padding：和卷积类似，可以取'VALID' 或者'SAME'  返回一个Tensor，类型不变，shape仍然是[batch, height, width, channels]这种形式'''  #### 卷积层2  W2 = tf.Variable(tf.truncated\_normal(shape=[3, 3, 32, 64], dtype=tf.float32, stddev=1e-2))  conv2 = tf.nn.conv2d(pool1, W2, strides=[1, 1, 1, 1], padding="SAME")  bn2 = tf.layers.batch\_normalization(conv2, training=train\_sign)  relu2 = tf.nn.relu(bn2)  pool2 = tf.nn.max\_pool(relu2, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")  #### 卷积层3  W3 = tf.Variable(tf.truncated\_normal(shape=[3, 3, 64, 128], dtype=tf.float32, stddev=1e-1))  conv3 = tf.nn.conv2d(pool2, W3, strides=[1, 1, 1, 1], padding="SAME")  bn3 = tf.layers.batch\_normalization(conv3, training=train\_sign)  relu3 = tf.nn.relu(bn3)  pool3 = tf.nn.max\_pool(relu3, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")  # 全连接层  dense\_tmp = tf.reshape(pool3, shape=[-1, 2\*2\*128]) # 有 2\*2\*128 个列  print(dense\_tmp)  fc1 = tf.Variable(tf.truncated\_normal(shape=[2\*2\*128, 512], stddev=0.01))  # tf.matmul() 矩阵点乘  bn\_fc1 = tf.layers.batch\_normalization(tf.matmul(dense\_tmp, fc1), training=train\_sign)  dense1 = tf.nn.relu(bn\_fc1)  dropout1 = tf.nn.dropout(dense1, rember)  '''tf.nn.dropout()是tensorflow里面为了防止或减轻过拟合而使用的函数，它一般用在全连接层  Dropout就是在不同的训练过程中随机扔掉一部分神经元。也就是让某个神经元的激活值以一定的概率p，  让其停止工作，这次训练过程中不更新权值，也不参加神经网络的计算。但是它的权重得保留下来（只是  暂时不更新而已），因为下次样本输入时它可能又得工作了'''  # 全连接层2  fc2 = tf.Variable(tf.truncated\_normal(shape=[512, 10], stddev=0.01))  out = tf.matmul(dropout1, fc2)  print(out)  cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=out, labels=y\_lab))  optimizer = tf.train.AdamOptimizer(0.01).minimize(cost)  dr = cifar\_reader.Cifar10DataReader(cifar\_folder="C:/Users/Administrator/cifar-10-batches-py/")  # 测试网络  correct\_pred = tf.equal(tf.argmax(out, 1), tf.argmax(y\_lab, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))  # 初始化所有的共享变量  init = tf.initialize\_all\_variables()  saver = tf.train.Saver()  # 开启一个训练  with tf.Session() as sess:  sess.run(init)  step = 1  while s \* batch\_size < train\_iter:  s += 1  batch\_xs, batch\_ys = dr.next\_train\_data(batch\_size)  # 获取批数据,计算精度, 损失值  opt, acc, loss = sess.run([optimizer, accuracy, cost],feed\_dict={input: batch\_xs, y\_lab: batch\_ys, rember: 0.6, train\_sign: True})  if s % show\_s == 0:  print ("训练进度 " + str(s\*batch\_size)+ " 张" + ", 损失值 = " + "{:.6f}".format(loss) + ", 训练准确率 = " + "{:.5f}".format(acc))  print ("训练完成。")  num\_examples = 10000  d,l = dr.next\_test\_data(num\_examples)  print ("测试集准确率:", sess.run(accuracy, feed\_dict={input: d, y\_lab: l, rember: 1.0, train\_sign: True}))  saver.save(sess, "model\_tmp/cifar10\_demo.ckpt") |