西北大学

大数据读书报告

基于 **KNN** 和利用 **TensorFlow** 框架分别对手写数字识别的实现

姓　　名　罗衍潮

学　　号　**201731913**

院　　系　信息科学与技术学院

专　　业　计算机技术

目录

1 基于 KNN 对手写数字识别的实现 1
1.1 KNN 算法的思想 1
1.2 数据准备 1
1.3 数据处理 2
1.4 运行环境 2
1.5 测试 2
1.6 运行结果 3
2 基于 TensorFlow 框架对手写数字识别的实现 4
2.1 算法的思想 4
2.2 数据准备 4
2.3 数据处理 4
2.4 递归下降、学习率 5
2.5 优化器的选择 5
2.6 运行环境 7
2.7 测试 7
2.6 运行结果 9
3 总结 9

1 基于 KNN 对手写数字识别的实现

1.1 KNN 算法的思想

对未知类别属性的数据集中的每个点依次执行以下操作：

1) **计算已知类别数据集中的点与当前点之间的距离**
2) **按照距离升序排列**
3) **选取与当前点距离最小的 K 个点**
4) **确定前 K 个点所在类别的出现频率**
5) **返回前 K 个点出现频率最高的类别当做当前点的预测分类**

1.2 数据准备

2000 个训练数据，其中 0-9 十个数字，每个数字 200 个训练数据集，900 个测试数据,训练数据放在 trainingDigits 文件夹下,测试数据放在 testDigits 文件夹下，数据形式如下：

0_1.txt 表示数字为 0 的第一个训练数据

0_10.txt 表示数字为 0 的第 10 个训练数据

......

0_99.txt 表示数字为 0 的第 99 个训练数据

0_1.txt	0_10.txt	1_100.txt
00000000000001100000000000000000 00000000000011111100000000000000 00000000000111111111000000000000 00000000011111111111000000000000 00000001111111111111100000000000 00000000111111100011110000000000 00000001111110000001110000000000 00000001111110000001110000000000 00000011111100000001110000000000 00000011111100000001111000000000 00000011111100000000011100000000 00000011111100000000011100000000 00000011111000000000001110000000 00000011111000000000001110000000 00000001111100000000000111000000 00000001111100000000000111000000 00000001111100000000000111000000 00000011111000000000000111000000 00000011111000000000000111000000 00000000111100000000000011100000 00000000111100000000000111100000 00000000111100000000000111100000 00000000111100000000001111100000 00000000011110000000000111110000 00000000011111000000001111100000 00000000011111000000011111100000 00000000011111000000111111000000 00000000011111100011111111000000 00000000000111111111111110000000 00000000000111111111111100000000 00000000000011111111110000000000 00000000000000111110000000000000	00000000000000110000000000000000 00000000000001111100000000000000 00000000000011111110000000000000 00000000101111111110000000000000 00000000111111111111000000000000 00000000111111111111100000000000 00000001111111111111110000000000 00000001111111100011111000000000 00000011111111000011111000000000 00000011111111000001111100000000 00000011111110000000111100000000 00000011111100000000111100000000 00000011111100000000111110000000 00000001111110000000011111000000 00000011111100000000001110000000 00000011111100000000001110000000 00000011111000000000001110000000 00000001111000000000001110000000 00000001111000000000001110000000 00000011111000000000001110000000 00000000111100000000000111000000 00000000111100000000000111000000 00000000111110000000001110000000 00000000111110000000011110000000 00000000111111000000111110000000 00000000011111110111111100000000 00000000000111111111111100000000 00000000000011111111111000000000 00000000000011111111110000000000 00000000000001111111100000000000 00000000000000111111000000000000	00000000000000000011111000000000 00000000000000000111111000000000 00000000000000000111111000000000 00000000000000011111111000000000 00000000000000011111111000000000 00000000000000011111111000000000 00000000000000011111111000000000 00000000000000111111111000000000 00000000000001111111111000000000 00000000000001111111111000000000 00000000000011111111111000000000 00000000000111111111111000000000 00000000011111111111110000000000 00000000111111111111110000000000 00000001111111111111110000000000 00000011111111111111110000000000 00000011111111011111110000000000 00000111111110011111110000000000 00000011111100011111110000000000 00000011111000011111110000000000 00000001110000011111110000000000 00000000000000111111000000000000 00000000000000111111000000000000 00000000000000011111100000000000 00000000000000011111100000000000 00000000000000011111100000000000 00000000000000001111110000000000 00000000000000001111111100000000 00000000000000000111111111000000 00000000000000000111111110000000 00000000000000000011111100000000
数字 0	数字 0	数字 1

1.3 数据处理

目录在 trainingDigits 包含了 2000 个例子：每个数字大约有 200 个，测试集有 900 个。

首先，将图像格式化处理为一个向量，把 32X32 转换成 1X1024 的向量

```
def img2vector(filename):
    returnVect=zeros((1,1024))
    fr=open(filename)
    for i in range(32):
        linestr=fr.readline()
        for j in range(32):
            returnVect[0,32*i+j]=int(linestr[j])
    return returnVect
```

1.4 运行环境

Python3.6+Windows10+Eclipse Pydev

1.5 测试

分类代码如下：

```
def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    #计算距离
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    #排序
    sortedDistIndicies = distances.argsort()
    classCount={}
    #选取前 K 个，输出概率最大的一个
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount)
    return sortedClassCount[0]
```

测试代码如下：
参数　K=3

```
def handwritingClassTest():
    hwLabels=[]
    trainingFieList=listdir('trainingDigits')
    m=len(trainingFieList)
    trainingMat=zeros((m,1024))
    for i in range(m):
        fileNameStr=trainingFieList[i]
        fileStr=fileNameStr.split('.')[0]
        classNumStr=int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:]=img2vector('trainingDigits/%s'%fileNameStr)
    testFileList=listdir('testDigits')
    errorCount=0.0
    mTest=len(testFileList)
    for i in range(mTest):
        fileNameStr=testFileList[i]
        fileStr=fileNameStr.split('.')[0]
        classNumStr=int(fileStr.split('_')[0])
        vectorUnderTest=img2vector('testDigits/%s'%fileNameStr)
        classifierResult=classify0(vectorUnderTest,trainingMat, hwLabels, 3)
        print("the classifier came back with: %d, the real answer is: %d
"%(classifierResult,classNumStr))
        if classifierResult!=classNumStr: errorCount+=1
    print("\n the total number of errors is : %d"%errorCount)
print("\n the total error rate is : %f"%(errorCount/float(mTest)))
```

1.6 运行结果

```
the classifier came back with: 1, the real answer is: 1
......
the classifier came back with: 9, the real answer is: 9
......
the classifier came back with: 4, the real answer is: 9
the total number of errors is : 31
the total error rate is : 0.032770
```

2 基于 TensorFlow 框架对手写数字识别的实现

2.1 算法的思想

1) 将要识别的图片转为灰度图，并且转化为 28*28 矩阵（单通道，每个像素范围 0-255，0 为黑色，255 为白色，这一点与 MNIST 中的正好相反）
2) 将 28*28 的矩阵转换成 1 维矩阵（也就是把第 2,3,4,5....行矩阵纷纷接入到第一行的后面）
3) 用一个 1*10 的向量代表标签，也就是这个数字到底是几，举个例子数字 1 对应的矩阵就是[0,1,0,0,0,0,0,0,0,0]
4) softmax 回归预测图片是哪个数字的概率
5) 用交叉熵和梯度下降法训练参数

2.2 数据准备

train-images-idx3-ubyte.gz: 训练集-图片，6w
train-labels-idx1-ubyte.gz: 训练集-标签，6w
t10k-images-idx3-ubyte.gz: 测试集-图片，1w
t10k-labels-idx1-ubyte.gz: 测试集-标签，1w

2.3 数据处理

对于识别 mnist 图片而言。
输入：784（=28X28）向量
输出：10（概率向量，概率最大的位置，就是预测的数字）

损失函数：即评价模型函数，评估网络模型的好坏，值越大，表示模型越差，反之，越好。常见的有交叉熵

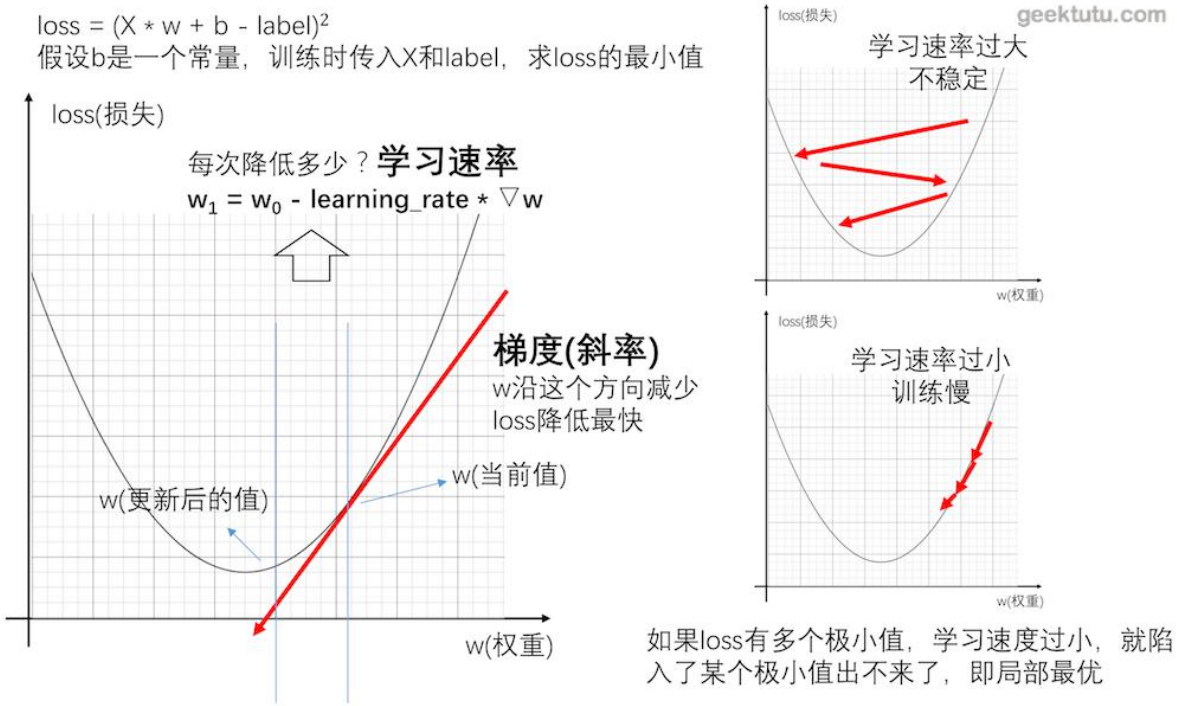
我们可以将网络理解为一个函数，回归模型，其实是希望对这个函数进行拟合。 比如定义模型为 Y = X * w + b，对应的损失

loss = (Y - labal)^2 = -(X * w - b - label)^2

这里损失函数用方差计算，这个函数是关于 w 和 b 的二次函数，所以神经网络训练的目的是找到 w 和 b，使得 loss 最小。

可以通过不断地传入 X 和 label 的值，来修正 w 和 b，使得最终得到的 Y 与 label 的 loss 最小。这个训练的过程，可以采用梯度下降的方法。通过梯度下降，找到最快的方向，调整 w 和 b 值，使得 w * X + b 的值越来越接近 label。

2.4 递归下降、学习率



学习率

简单说，梯度即一个函数的斜率，找到函数的斜率，其实就知道了 w 和 b 的值往哪个方向调整，能够让函数值（loss）降低得最快。这个数，神经网络中称之为学习速率。学习速率调得太低，训练速度会很慢，学习速率调得过高，每次迭代波动会很大。

2.5 优化器的选择

大多数**机器学习任务**就是**最小化损失**，在损失定义的情况下，后面的工作就交给优化器。 因为**深度学习**常见的是对于**梯度的优**化，也就是说，优化器最后其实就是各种对于**梯度下降算法的优化**。详细方法见源码，源码如下：

2.5.1 tf.train.Optimizer

优化器（optimizers）类的基类。这个类定义了在训练模型的时候添加一个操作的 API。

2.5.2 tf.train.GradientDescentOptimizer

这个类是实现梯度下降算法的优化器。(结合理论可以看到，这个构造函数需要的一个学习率就行了)
__init__(learning_rate, use_locking=False,name=’ GradientDescent’)

2.5.3 tf.train.AdagradOptimizer

__init__(learning_rate,initial_accumulator_value=0.1,
use_locking=False, name=’ Adagrad’)

2.5.4 tf.train.AdadeltaOptimizer

实现了 Adadelta 算法的优化器，是上面的 Adagrad 算法改进版本
构造函数：
__init__(learning_rate=0.001,rho=0.95,epsilon=1e-08,
use_locking=False, name=’Adadelta’)

2.5.5 tf.train.MomentumOptimizer

tf.train.MomentumOptimizer(learning_rate, 0.9)
Momentum 可以使 SGD 不至于陷入局部鞍点震荡，同时起到一定加速作用。
Momentum 最开始有可能会偏离较远(overshooting the target)，但是通常会慢慢矫正回来。

2.5.6 tf.train.RMSPropOptimizer

tf.train.RMSPropOptimizer(learning_rate, 0.9)

2.5.7 tf.train.AdamOptimizer

__init__(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08,
use_locking=False, name=’ Adam’)

2.6 运行环境

Python3.6+Windows10+Eclipse Pydev

2.7 测试

构造网络模型 Model.py

```
#coding:utf-8
import tensorflow as tf
class NetWork(object):
    def __init__(self):
        # 学习速率，一般在 0.00001 - 0.5 之间
        self.learning_rate = 0.001
        # 输入张量 28 * 28 = 784 个像素的图片一维向量
        self.x = tf.placeholder(tf.float32, [None, 784])
        # 标签值，即图像对应的结果，如果对应数字是 8，则对应 label 是 [0,0,0,0,0,0,0,0,1,0]
        # 这种方式称为 one-hot 编码
        # 标签是一个长度为 10 的一维向量，值最大的下标即图片上写的数字
        self.label = tf.placeholder(tf.float32, [None, 10])
        # 权重，初始化全 0
        self.w = tf.Variable(tf.zeros([784, 10]))
        # 偏置 bias， 初始化全 0
        self.b = tf.Variable(tf.zeros([10]))
        # 输出 y = softmax(X * w + b)
        self.y = tf.nn.softmax(tf.matmul(self.x, self.w) + self.b)
        # 损失，即交叉熵，最常用的计算标签(label)与输出(y)之间差别的方法
        self.loss = -tf.reduce_sum(self.label * tf.log(self.y + 1e-10))
        # 反向传播，采用梯度下降的方法。调整 w 与 b，使得损失(loss)最小
        # loss 越小，那么计算出来的 y 值与 标签(label)值越接近，准确率越高

       #self.train =
    tf.train.GradientDescentOptimizer(self.learning_rate).minimize(se
    lf.loss)
        self.train=tf.train.AdamOptimizer(self.learning_rate, beta1=0.9,
beta2=0.999, epsilon=1e-08).minimize(self.loss)
        #tf.train.AdadeltaOptimizer(self.learning_rate, rho=0.95,
epsilon=1e-08).minimize(self.loss)
        # 以下代码验证正确率时使用
        # argmax 返回最大值的下标，最大值的下标即答案
        # 例如 [0,0,0,0.9,0,0.1,0,0,0,0] 代表数字 3
        predict = tf.equal(tf.argmax(self.label, 1), tf.argmax(self.y, 1))
        # predict -> [true, true, true, false, false, true]
        # reduce_mean 即求 predict 的平均数 即 正确个数 / 总数，即正确率
        self.accuracy = tf.reduce_mean(tf.cast(predict, "float"))
```

训练网络模型 Train.py

```
import NetWork
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
class Train(object):
    def __init__(self):
        self.net=NetWork()
        # 初始化 session
        # Network() 只是构造了一张计算图，计算需要放到会话(session)中
        self.sess = tf.Session()
        # 初始化变量
        self.sess.run(tf.global_variables_initializer())
        # 读取训练和测试数据，这是 tensorflow 库自带的，不存在训练集会自动下载
        # 项目目录下已经下载好，删掉后，重新运行代码会自动下载
        self.data = input_data.read_data_sets('data_set', one_hot=True)
    def train(self):
        # batch_size 是指每次迭代训练，传入训练的图片张数。
        # 数据集小，可以使用全数据集，数据大的情况下，
        # 为了提高训练速度，用随机抽取的 n 张图片来训练，效果与全数据集相近
        batch_size = 64
        # 总的训练次数
        train_step = 2000
        # 开始训练
        for i in range(train_step):
            # 从数据集中获取 输入和标签(也就是答案)
            x, label = self.data.train.next_batch(batch_size)
            # 每次计算 train，更新整个网络
            # loss 只是为了看到损失的大小，方便打印
            _, loss = self.sess.run([self.net.train, self.net.loss],
                      feed_dict={self.net.x: x, self.net.label: label})
            # 打印 loss，训练过程中将会看到，loss 有变小的趋势
            # 代表随着训练的进行，网络识别图像的能力提高
            # 但是由于网络规模较小，后期没有明显下降，而是有明显波动
            if (i + 1) % 10 == 0:
                print('第%5d 步，当前 loss：%.2f' % (i + 1, loss))
    def calculate_accuracy(self):
        test_x = self.data.test.images
        test_label = self.data.test.labels
        # 注意：与训练不同的是，并没有计算 self.net.
        # 只计算了 accuracy 这个张量，所以不会更新网络
        # 最终准确率约为 0.91
        accuracy = self.sess.run(self.net.accuracy,
           feed_dict={self.net.x: test_x, self.net.label: test_label})
        print("准确率：%.2f，共测试了%d 张图片 " % (accuracy, len(test_label)))
```

```
    测试网络模型__init__.py
import Train
if __name__=="__main__":
    app=Train()
    app.train()
    app.calculate_accuracy()
```

2.6 运行结果

选择 AdamOptimizer 优化器的运行结果如下：

beta1=0.9,beta2=0.999, epsilon=1e-08	beta1=0.99,beta2=0.9999, epsilon=1e-08
第 10 步，当前 loss：117.76 ..... 第 1960 步，当前 loss：14.80 ..... 第 2000 步，当前 loss：25.67	第 10 步，当前 loss：130.88 ..... 1960 步，当前 loss：24.65 ..... 第 2000 步，当前 loss：23.57
准确率：0.91，测试了 10000 张图片	准确率：0.92，测试了 10000 张图片

3 总结

本报告分析在不同的算法，对手写数字识别的实现，KNN 算法是比较经典的分类算法，尽管算法思想简单，但在有些领域还时有着不错的分类效果，现在，Google 以经开发出 TensorFlow 框架，我们可以借助框架，进一步缩短开发编码的时间、简化工作量。利用数学上的知识，对算法进行调参优化，通过调参优化过程 ，寻找到最好的参数，更好的将数据泛化，从而，寻找到一个更好的分类器 。

编写报告的同时，学到了很多知识，对框架的进一步理解，优化器的选择，调参的步骤过程 ，这给以后的学习工作提供了良好的经验。