山东大学 软件 学院

机器学习 课程实验报告

|  |  |  |
| --- | --- | --- |
| 学号：201800301153 | 姓名： 傅显坤 | 班级： 1班 |
| 实验日期： 2020.11.20 | | |
| 实验题目：**神经网络学习**  实验给出的数据:  CIFAR-10数据集，数据集中包含 50000 张训练样本，10000 张测试样本，可将训练样本划分为49000 张样本的训练集和1000 张样本的验证集，测试集可只取1000 张测试样本。其中每个样本都是 32×32 像素 的彩色照片，每个像素点包括RGB三个数值，数值范围0 ~ 255，所有照片分属10个不同的类别。  实验的问题  (1) 用神经网络对给定的数据集进行分类，画出loss图，给出在测试集上的精确度；  (2) 不能使用 TensorFlow 等框架，也不能使用库函数，所有算法都要自己实现；  (3) 神经网络结构图如下图所示：    整个神经网络包括 3 层——输入层，隐藏层，输出层。输入层有 32\*32\*3 个神经元，隐藏层有 50个神经元，输出层有 10 个神经元（对应 10 个类别）。  (4) 附加：可以试着修改隐藏层神经元数，层数，学习率，正则化权重等参数探究参数对实验结果的影响。 | | |
| 软件环境：  MacOS Catalina  Python3.0  IDE：PyCharm | | |
| 实验步骤与内容：  1.数据导入  首先加载训练集的文件——cifar-10-batches-py，将每个图片数据格式化为3072\*1的numpy数组，标签数据格式化为10\*1的numpy数组。  训练集图片是50000\*3072的numpy数组，标签是10000\*10的numpy数组（那个值为1表示它属于哪一类）。   | **名称** | **作用** | | --- | --- | | b’data’ | 是一个10000x3072的array，每一行的元素组成了一个32x32的3通道图片，共10000张 | | b’labels’ | 一个长度为10000的list，对应包含data中每一张图片的label |   2.模型分析  每个样本是32×32的像素点，每个像素点有RGB三种取值，数值范围0 ~ 255。所以设计神经网络时的输入节点为32\*32\*3=3072个节点。  因为所有照片分属10个不同的类别，则设置10个输出神经元。  一开始学习时，隐藏层设置为一层，神经元个数为50个。  3.学习过程  初始化权重w和偏执因子b：  经过查阅资料选择每层的w选择：均值为0，标准差为该层输出神经元的个数的-0.5次方的正态分布，b选择：在0，1之间的随机数  3.1）正向学习  公式一：  公式二：  公式三（激活函数选择）：  通过将公式一中的结果带入到公式二中获得预测值，激活函数选择公式三。  3.2）反向学习  公式四(损失函数)：  公式五（定义的误差\_输出层和中间层）：  公式六(激活函数的导数)：  公式七（权值的更新）：  公式八(偏执因子的更新)：  通过以上五个公式，反向学习，获得最后的权值w和偏执因子b。  3.3）重复正向学习和反向学习  4.画出loss图以及给出测试集上的精准度  Loss图：loss是训练集的损失值，val\_loss是测试集的损失值  通过训练集中前49000图片计算loss和后1000张图片作为验证集计算val\_loss估计模型的拟合性。  测试集上的精准度：取验证集中1000张图片，放入模型中，看正确分类的个数占总的比例。 | | |
| 实验结果：  实验感悟：  通过这次实验更加深入的了解了神经网络算法的运行过程，尤其是反向学习的过程，这个在课堂上的理解不是很到位，导致课下查了很多资料才弄清楚，反向学习的具体过程。  学会了通过loss图来诊断模型。通常数据集会被划分成三部分，训练集（training dataset）、验证集（validation dataset）、测试集（test dataset）。我们在训练模型时也经常会根据训练集的loss和验证集loss来诊断模型，从而期望能够优化参数训练处一个更好的模型。  实验中问题，不能将验证集放入模型中学习，否则会导致val\_loss很小，这就是老师上课说的容易出现的“监守自盗”问题，学习过这些数据，肯定导致loss很低，这是我犯过的一个错误。  通过不断运行模型，我观察到以下几种情况：   1. 不断改变学习率：   通过改变学习率，观察模型的变化：  当学习率从0.01不断增加时，学习率为0.01，0.05和0.08的模型表现较好，可以看出val\_loss已经趋近于某个值附近，说明模型已经学习完毕。但是当学习率为0.15时出现了明显的过拟合现象（over fitting），通过实验理解了over fitting的实际意义。由于0.05时模型的测试准确率最高，我们最终选择学习率为0.05.   1. 不断改变中间神经元的个数：   当选择学习率为0.05时，将中间神经元的个数由50变为100个再到150，250个，观察模型发现当中间神经元为150个时，模型上的准确率最高达到了0.542，当250个时准确率略有下降，所以判断神经元的个数一味增加也不一定导致准确率的上升（由于时间问题我就跑到250个神经元，每次跑算法的时间太长了）。  3）改变隐藏层的层数：  使隐藏层的个数为两层，可以看到不管中间神经元的个数是50，30还是100，50时，loss图都出现了明显的抖动现象。通过查阅资料得出可能是因为训练数据集的规模太小，不能使复杂的神经网络收敛。  4）改变学习的次数  当学习的次数由50到80时可以看出虽train\_loss还在缓慢下降，但是val\_loss已经趋于收敛，而且测试集上的准确率对比与50步反而下降了，说明模型的val\_loss已经收敛无需更多的步数了。  下面是实验结果数据：   1. 当神经网络只有两层，中间神经元为50个时    1. 当学习率为0.01时：   Loss    测试集上的准确率    1.2）当学习率是0.05时：  Loss图  测试集上的精准度：    1.3）当学习率是0.08时：    测试集上的精准度    1.4）当学习率是0.15时：    测试集上的精准度    2.当神经网络只有两层，学习率是0.05，改变中间神经元数：  2.1）中间神经元为100个时    测试集上的准确率为    2.2）中间神经元为150个时    测试集上的准确率    2.3）当神经网络只有两层，中间神经元为250个时    测试集上的准确率为：    3. 当神经网络有三层，学习率是0.05  3.1）中间神经元为50个和30个时    测试集上的准确率：    3.2）中间神经元为100和50个时：    测试集上的准确率为：    4.当更改实验步数  80步    测试集上的准确率    重要代码：  1）学习过程  # 训练函数  def update(self, input\_nodes, targets):     inputs = numpy.array(input\_nodes, ndmin=2).T     targets = numpy.array(targets, ndmin=2).T  ​     # 前向传播——关键在于计算出每层的损失     # 定义输出值列表（outputs[0]为输入值）     self.outputs = []     self.outputs.append(inputs)     # 将神经网络的每一层计算输出值作为输入放入激活函数中，并保存到outputs列表中     for i in range(self.numNeuronLayers):         # 每个神经元的输入信号z，由参数w、b，和前一层的输出信号a决定         temp\_inputs = numpy.dot(self.weight[i], inputs) + self.bias[i]         temp\_outputs = self.activation\_function(temp\_inputs)         inputs = temp\_outputs         self.outputs.append(temp\_outputs)     # 计算每层的损失     self.output\_errors = []     for i in range(self.numNeuronLayers):         # 输出层的误差=目标值-输出值         if i == 0:             self.output\_errors.append(targets - self.outputs[-1])         # 隐藏层的误差=（当前隐藏层与下一层之间的权值矩阵）的转置与下一层的误差矩阵的乘积         else:             self.output\_errors.append(numpy.dot((self.weight[self.numNeuronLayers - i]).T,                                                 self.output\_errors[i - 1]))     # 反向传播     for i in range(self.numNeuronLayers):         # 权值更新规则为之前 新权值=权值+学习率\*误差\*激活函数的导数\*上一层输出         # 偏移量b的更新规则 新偏执因子=偏执因子+学习率\*误差\*激活函数的导数         # f(x)\*（1-f(x)）即为激活函数f(x)的导函数         self.weight[self.numNeuronLayers - i - 1] += \             self.learningrate \* numpy.dot(                 self.output\_errors[i] \* self.outputs[-1 - i] \* (1 - self.outputs[-1 - i]),                 self.outputs[-2 - i].T)         self.bias[self.numNeuronLayers - i - 1] += self.learningrate \* (                 self.output\_errors[i] \* self.outputs[-1 - i] \* (1 - self.outputs[-1 - i]))  2）loss图的绘制：  train\_loss = [] test\_loss = [] cycles = 80 for i in range(cycles):    print(i)    for i in range(len(images\_data)):        n.update(images\_data[i], labels[i])    print("train\_loss=" + str((len(images\_data) - getTrueNumber(images\_data, labels, n)) / len(images\_data)))    train\_loss.append((len(images\_data) - getTrueNumber(images\_data, labels, n)) / len(images\_data))    print("test\_loss=" + str(       (len(verification\_data) - getTrueNumber(verification\_data, verification\_labels, n)) / len(            verification\_data)))    test\_loss.append((len(verification\_data) - getTrueNumber(verification\_data, verification\_labels, n)) / len(        verification\_data)) x = [] for i in range(cycles):    x.append(i + 1) plt.plot(x, train\_loss, color="r", linestyle="--", marker="\*", linewidth=2.0, label='loss') plt.plot(x, test\_loss, color="b", linewidth=2.0, label='val\_loss') plt.legend(loc='upper left', bbox\_to\_anchor=(0.2, 0.95)) plt.title("LOSS") plt.show() | | |