计算机科学与技术学院神经网络与深度学习课程实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：神经网络的基本训练 | | 学号：201900150221 |
| 日期：10.7 | 班级： 19智能 | 姓名： 张进华 |
| Email：zjh15117117428@163.com | | |
| 实验目的：  掌握基本的神经网络调整技巧并尝试，改进深度神经网络：超参数调整、正则化和优化，共三个子任务：初始化、梯度检查和优化 | | |
| 实验软件和硬件环境：  Visual studio Code python 3.9.7  Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz | | |
| 实验原理和方法  1.完成三个初始化，梯度计算，和最优化三个文件，分别实现了神经网络中最常用的集中优化神经网络的方式。  2.数据是直接用的是 sklearn 的专有的数据。每个都有对应的 py 文件，这个文件里面的东西不需要进行修改， 直接使用即可。  3.初始化，用于初始化权重，即神经元最开始的 w 的值，方便计算出不同的东西。  4.梯度检查，用于验证反向传播是否有效，正确，也就是在训练模型的时候，是否对 dw 和 db 的值更新正确。  5.优化，是用于加快函数计算速度，比如调整学习率之类的操作来进行。 | | |
| 实验步骤  **步骤一 Initialization**  实现权重初始化  **1.Neural Network model**  Zeros 初始化-- 在输入参数中设置 initialization = "zeros"  随机初始化-- 在输入参数中设置initialization = "random" 将权重初始化为大的随机值  He 初始化 -- 在输入参数中设置initialization = "he" 将权重初始化为根据 He 等人在 2015 年发表的论文缩放的随机值  **2 .Zero initialization**  实现函数以将所有参数初始化为零,不能很好地工作，因无法“破坏对称性”    使用这些参数来训练模型    图中我们可以看到学习率一直没有变化，结论为模型根本没有学习，查看预测结果：    可见算法性功能比较差，查看预测和决策边界    分类失败，该模型预测每个都为0。零初始化都会导致神经网络无法打破对称性，最终导致的结果就是无论网络有多少层，最终只能得到和Logistic函数相同的效果  **3. Random initialization**  为了打破对称性，可以随机地把参数赋值。在随机初始化之后，每个神经元可以开始学习其输入的不同功能    在得到参数后运行    由图可以看到模型比初始化全部为0效果要好一点，查看分类结果    **4. He initialization**  使用给定公式初始化参数    使用得到的参数进行训练预测    由图可以看到效果比随机设置的参数好，误差越来越小，查看分类效果    总结：不同的初始化方法可能导致性能最终不同，随机初始化有助于打破对称，使得不同隐藏层的单元可以学习到不同的参数。初始化时，初始值不宜过大。He初始化搭配ReLU激活函数可以得到不错的效果  **步骤二 Gradient Checking**  梯度计算公式如下    **1.1-dimensional gradient checking**  由公式可以得到    实现反向传播：    进行梯度检查，梯度检查的步骤如下：    接下来，计算梯度的反向传播值，最后计算误差，当difference小于1 0 − 7 时，我们通常认为我们计算的结果是正确的    根据公式实现代码如下：    进行测试结果显示正确    **2. N-dimensional gradient checking**  实现梯度检查的伪代码如下：  根据公式实现代码如下：    进行测试结果显示与官方文档中的一致：    结果说明backward\_propagation\_n 代码中似乎有错，通过实现梯度检查，检查dW2 和db1纠正错误，修改代码如下：    已修复完成后，重新运行梯度检查，测试结果显示正确    **步骤三 Optimization**  在深度学习中，如果数据集没有足够大的话，可能会导致一些过拟合的问题。过拟合导致的结果就是在训练集上有着很高的精确度，但是在遇到新的样本时，精确度下降会很严重。为了避免过拟合的问题，使用正则化的方式   1. **Gradient Descent**   梯度下降参数更新规则：    根据公式实现代码如下：    **2. Mini-Batch Gradient descent**  首先从训练集（X，Y）构建小批量，创建训练集 (X, Y) 的混洗版本， X 和 Y 的每一列代表一个训练示例。将混洗后的 (X, Y) 分成大小为“mini\_batch\_size”（此处为 64）的小批量，更新公式如下：    实现代码如下：    **3.Momentum**  小批量梯度下降只看到一个样本子集后进行参数更新，更新的方向有一定的差异，所以小批量梯度下降所采取的路径会“振荡”向收敛。 使用动量可以减少这些振荡。Momentum 考虑了过去的梯度来平滑更新。 将先前梯度的“方向”存储在变量 v 中，是指先前步骤梯度的指数加权平均值，也可以将 v 视为滚下山坡的球的“速度”，根据坡度/坡度的方向增加速度（和动量）  更新公式如下：    实现代码如下：    接下来使用动量实现参数更新。 动量更新规则是：    其中 L 是层数，beta是动量，alpha是学习率，实现代码如下：    速度初始化为零，算法将进行几次迭代以“建立”速度并开始采取更大的步骤。如果beta = 0，那么这就是没有动量的标准梯度下降。动量 beta越大，更新越平滑，因为我们越多地考虑过去的梯度。但是如果 beta太大，它也会过度平滑更新。  **4.Adam**  Adam结合了 RMSProp和 Momentum 的想法，计算过去梯度的指数加权平均值，并将其存储在变量v（在偏差校正之前）和 v^{corrected}（在偏差校正之前）。计算过去梯度平方的指数加权平均值，并将其存储在变量 s（在偏差校正之前）和 s^{corrected}（在偏差校正之前）。根据组合来自“1”和“2”的信息在一个方向上更新参数，更新规则如下：    t 计算 Adam 采取的步数，L 是层数，beta\_1和 beta\_2是控制两个指数加权平均值的超参数，alpha$ 是学习率，varepsilon$ 是一个非常小的数字，以避免被零除，实现代码如下：    现在，使用 Adam 实现参数更新，一般的更新规则如下：    实现代码如下：    **5.  Model with different optimization algorithms**  使用“moons”数据集来测试不同的优化方法，加载数据如下：    然后使用 3 种优化方法中的每一种运行 3 层神经网络  **5.1 - Mini-batch Gradient descent**  查看模型如何使用小批量梯度下降，实现效果如图所示：    由图可知该模型准确率很高，Accuracy = 0.796,效果很好。  **5.2 - Mini-batch gradient descent with momentum**  查看模型如何处理动量，由于例子比较简单，使用momentum的收益很小，实现效果如图所示：    可见该模型得到的效果与梯度下降效果差不多  **5.3 - Mini-batch with Adam mode**  查看模型如何处理 Adam，实现效果如图所示：    可见使用Adam模型得到模型准确率最高，Accuracy = 0.94,效果最好 | | |
| 结论分析与体会：  Momentum通常有效果，但考虑到小的学习率和简单的数据集，它的影响几乎可以忽略不计。 此外，对于优化算法来说，一些小批量比其他小批量更难。另一方面，Adam 明显优于小批量梯度下降和 Momentum，Adam 收敛得更快。  Adam 的一些优点包括：  - 内存需求相对较低（虽然高于梯度下降和带有动量的梯度下降）  - 通常即使很少调整超参数也能很好地工作（除了 alpha） | | |
| 就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟1－3道问答题：  这次实验总体上来说就是内容比较明确，实验过程中代码大多数情况下可以由给定公式推导出来，主要的难点在于不清楚python中对于列表、字典的运算，花费了点时间在网上学习。其实，最难的部分还是在优化部分，虽然指导书上有讲解，但是理解起来还是有些困难。 | | |