作业 3 神经网络模型

晏瑞然　87　PB19000196

2021 年 4 月 11 日

摘要

通过简单的神经网络 (Neural Networks) 模型，得到一个分类器，实现对昆虫以及一个 20 类点集的分类。

一、前言

1.1 神经网络介绍

模拟人类实际神经网络的数学方法问世以来，人们已慢慢习惯了把这种人工神经网络直接称为神经网络。神经网络在系统辨识、模式识别、智能控制等领域有着广泛而吸引人的前景，特别在智能控制中，人们对神经网络的自学习功能尤其感兴趣，并且把神经网络这一重要特点看作是解决自动控制中控制器适应能力这个难题的关键钥匙之一。

1.2 问题概述

有两组数据——训练集 (training set) 与测试集 (test set)，训练集包含元素的特征 (features) 与标签 (labels)，测试集只有元素的特征。构造一个神经网络并通过训练集训练神经网络模型，使之能够预测测试集中元素的标签。

二、相关工作

引用文献 [1]

三、问题分析

不妨设特征的个数为 m，需要分的类的个数为 n。所谓的分类，即找到一个 m 维到 n 维的映射 $f : R^m \to R^n$ ，通过该映射便能将特征映射成标签。

要通过训练集得到这个映射，首先的构造一个网络，每层的神经元由上一层所有神经元映射而来，第一层神经元就是特征，最后一层即为得到的标签。隐藏层的神经元代表着特征中的隐藏的特征，最终输出我们需要的隐藏特征，即元素标签。

神经元之间的映射首先想到的便是用线性组合，但这并不好，如果只有线性组合那么整个网络都只有特征的线性相关项，那么整个网络就没有意义只不过是一个线性映射，这在一些数据集非线性分类的情况下会失去意义，如一个分类标准是一个圆，圆内为 labe[0] 圆外为 label[1]，这种方法就会失效。所以我们需要在映射中加入非线性项，我们就在线性映射上加一个激活函数，激活函数的种类有很多，如 relu 函数，softmax 函数…通过这些非线性函数对网络加入非线性项，使得网络能够非类非线性模型。

网络的训练通过反向传播 BP(back propagation) 算法，不断修改网络层中的线性映射的系数，最终便能得到想要的映射。

四、建模的假设

4.1 假设 1

所给数据集是可分的，可以有噪声但不能不可分。

4.2 假设 2

所有数据不能有特征相同但标签不同的情况。

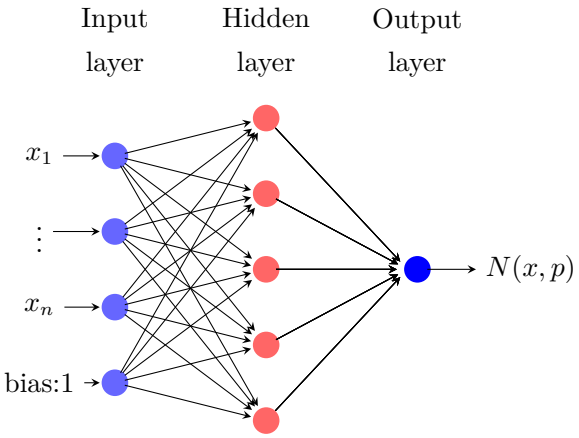
五、符号说明

表 1: **符号说明**

符号	说明
x_i	第 i 个特征向量
y_i	第 i 个输出
m	特征的维度
n	输出的维度
N	总数据量
x_{ic}	第 i 个特征向量的第 c 个特征
y_{ic}	1 表示第 i 个数据标签是 c，0 反之
p_{ic}	第 i 个输出是第 c 类的概率
X^k	第 k 层神经元组成的列向量
ω_{ij}^k	k 层网络 i 神经元到下层 j 神经元的权重
W^k	ω_{ij}^k 组成的系数矩阵
α	学习速率
b	bias 项
act	激活函数

六、数学模型建立

6.1 神经网络



神经网络结构图所示由输入层的线性组合得到第一层隐藏层，在隐藏层中加入激活函数引入非线性项，再通过同样的方法传递到后面的隐藏层，最终传递到输出层。即可得到需要的 m 到 n 维的映射。传播形式可写为矩阵形式：

$$X^{k+1} = act(W^k X^k + b)$$

其中 act 函数可以使用以下函数：

1) sigmoid 函数: $sigmoid(x) = \frac{1}{1+e^x}$

2) ReLU 函数: $ReLU(x) = max(0, x)$

3) tanh 函数: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

3) ELU 函数:

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

注意每次传递后要进行正规化，这样能很好的提高效率。最后，使用 softmax 函数进行归一化得到 y_{ic}:

$$softmax(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

6.2 反向传播 (BP)

有了网络的结构，我们还需要确定网络中的参数。首先通过随机的参数得到最初的结果，这个结果肯定是不好的，我们可以通过不同的损失函数计算预测结果与真实结果的误差然后最小化该损失函数，最小化的方法可以使用梯度下降。本文中的损失函数使用的是交叉熵函数：

$$J = -\frac{1}{N}\sum_i (\sum_{c=1}^{n} y_{ic} log(p_{ic}))$$

通过梯度下降，不断的更新 ω_{ij}^k:

$$\omega_{ij}^k = \omega_{ij}^k - \alpha \frac{\partial J}{\partial \omega_{ij}^k}$$

多次迭代后便可得到 J 的局部最优解。此过程便是神经网络的训练 (training) 过程。

七、结果

7.1 结果说明

通过神经网络得到 7.2 中结果，其中图片标签格式为：网络结构学习速率迭代次数准确率程序运行时间。如从 insets 数据集中得到的结果其中网络结构为 $2 \to 10 \to 6 \to 3$ ，学习速率为 0.02，迭代次数为 1000 次，准确率是 90%，运行了 4 秒则标签为——"2-10-6-3 0.02 1000 90 4"。图的 x,y 坐标分别为数据集的数据集的两个特征，点的颜色代表其种类。左上为数据集拟合结果，右上为测试集应有结果，左下为预测结果。

图 1 为昆虫数据集的测试结果。通过图 1 的比较可以看出 100 次迭代后误差仍较大；达到 1000 次迭代后效果还不错，时间也相对较少；10000 次迭代结果与 1000 次基本相同时间却大量增加。展示 loss 值后可以看出，该学习速率下 1000 次迭代就基本收敛，之后的迭代只会让 loss 不断震荡，无法得到更精确的解，这是因为此时迭代已接近收敛，梯度下降已经很难得到更高的精度了。相比与时间开销我们可以牺牲一些准确度，所以迭代次数选在 1000 左右即可。

图二比较了不同学习速率的 loss-迭代次数曲线，可以看出学习速率在 0.01 的时候收敛速度较慢，接近 1000 次迭代才勉强能看出收敛迹象。0.05 的学习速率表现的就很好。而 0.1 的学习速率可以看出虽然收敛速度很快 (400 次左右就收敛了)，但因为学习速率过高很难准确找到局部最优解，其解在最优解附近不停震荡，在图中的表现就是相邻迭代次数损失值上下跳跃。所以得出结论，学习速率大概在 0.05 左右即可。

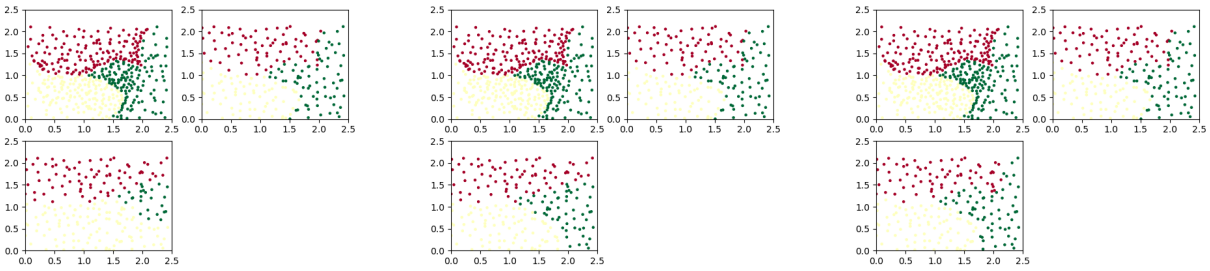
图三进行了不同网络结构的比较 (单隐藏层与多隐藏层的比较)。可以看到同样的迭代次数与学习速率多层网络表现的更好，准确率 90% 左右而单层只能达到 85%。

由上我们得到了 insets 数据集一组较好的参数: 采用 2-10-6-3 结构，learningrate 取 0.05, 迭代次数取 1000。用此参数我们进行有噪声的 Insets 训练集的测试。见图四。

通过与上相同的方法调整 points 数据的参数，图 5 展示了有噪声和无噪声的点集数据集的测试结果。使用的参数为：2-64-36-20 结构，0.05 学习速率，1000 次迭代。

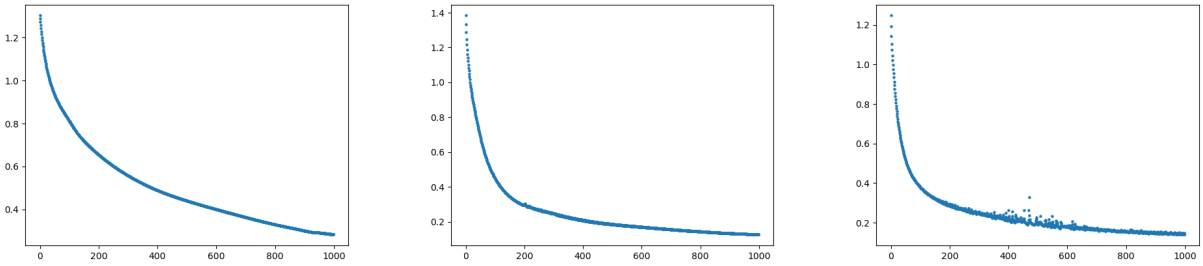
图六用不同激活函数收敛速度的对比。(数据集为 Points 数据集，都采用双隐藏层结构，学习速率为 0.05, 迭代次数截至到 1000 为止)。可以看出 relu 与 elu 函数收敛速度较快；tanh 收敛速度一般；而 sigmoid 收敛速度极慢，1000 次迭代都没有接近收敛，loss 还在 1.5 左右。

7.2 结果展示



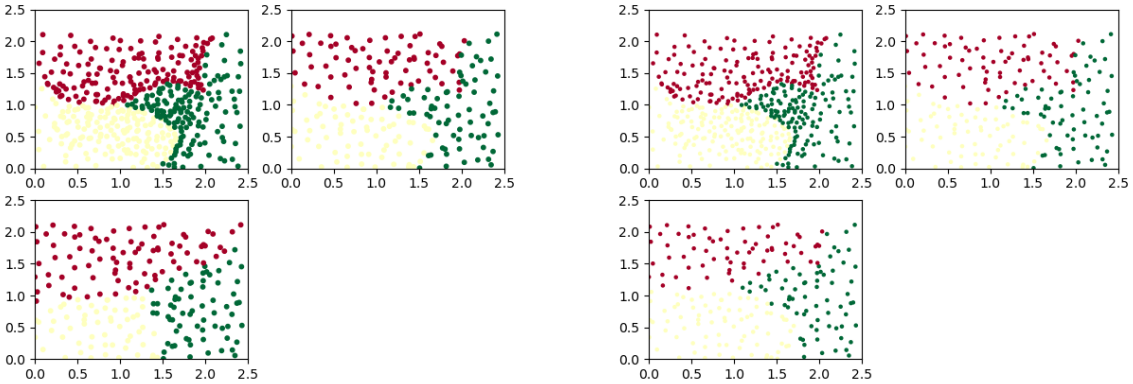
(a) 2-10-6-3 0.02 100 73.81 05.070578 (b) 2-10-6-3 0.02 1000 87.14 24.342106 (c) 2-10-6-3 0.02 10000 89.05 4:02.436815

图 1: 昆虫数据集不同迭代次数的比较



(a) learningrate=0.01 (b) learningrate=0.05 (c) learningrate=0.1

图 2: 昆虫数据集 2-10-6-3 网络结构 loss-迭代次数 图像



(a) 2-10-3 结构准确率 85.24% (b) 2-10-6-3 结构准确率 89.05%

图 3: 昆虫数据集 2-10-6-3 网络结构 loss-迭代次数 图像

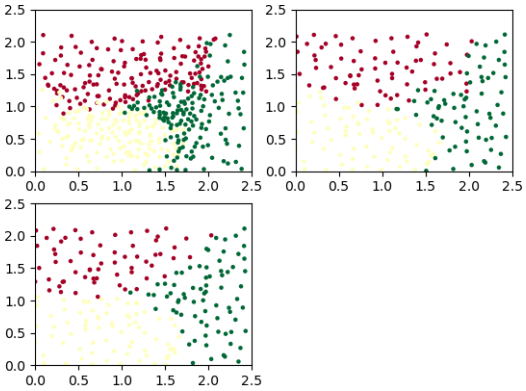
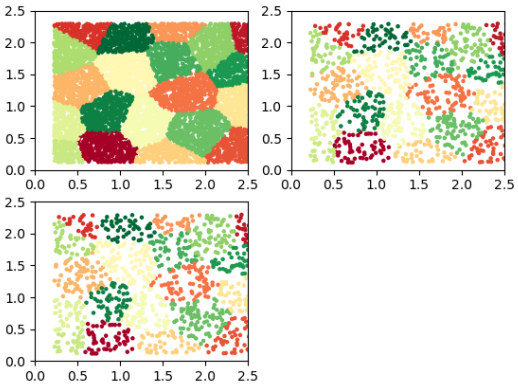
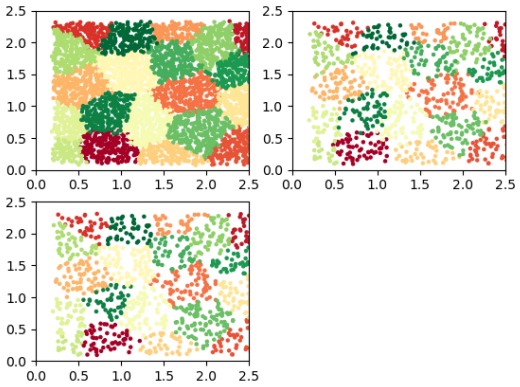


图 4: 有噪声 insets 数据集结果 准确率为 89.52%

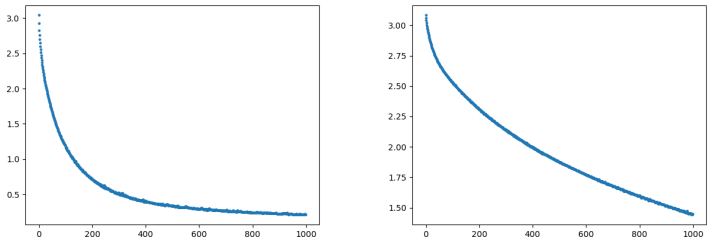


(a) 无噪声 Points 数据集测试结果准确率 94.21%

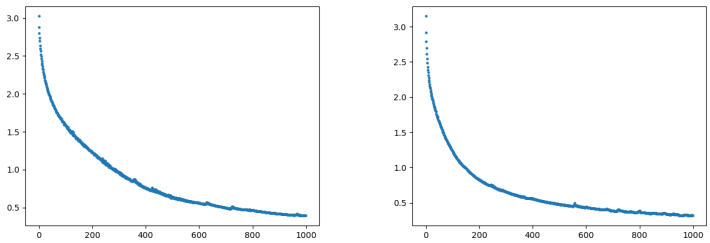


(b) 有噪声 Points 数据集测试结果准确率 91.61%

图 5: points 数据集测试结果图像



(a) 激活函数为 relu (b) 激活函数为 sigmoid



(c) 激活函数为 tanh (d) 激活函数为 elu

图 6: 不同激活函数收敛速度的比较

八、结论

由上述结果可知以下结论：

1) 不同的网络结构会影响结果，网络越复杂越容易发现数据中的隐藏规则，但相对的也会牺牲一些时间效率，我们需要找到两者之间的平衡点。一般像本次作业中一样的简单数据并不需要过于复杂的网络结构，1 到 2 层隐藏层即可。

2) 通过不同数量级的迭代次数对比，可以得到最佳的迭代次数，避免迭代次数过少还未收敛就结束了程序，同时也减少了时间的浪费，接近收敛时就结束迭代，不会进行过多的无用迭代。也可以从 loss-迭代次数图像中找到最佳迭代次数。

3) 学习速率过高会导致无法精确找到局部最优解，导致结果效果差。而学习速率过低会使得收敛时间变长，找到局部最优解会浪费大量时间，所以需要找到合适的学习速率平衡精确度与时间开销。

4) 不同激活函数对收敛速度有所影响，应选取合适的激活函数，ReLU 函数就是个很好的函数。

九、问题与优化

1) 网络传播过程中要进行正规化 (normalization), 这样能大幅提升梯度下降速率。经测试不进行正规化的复杂网络 (多层网络) 收敛速度极慢，基本需要 10000 到 100000 次迭代才能基本有效；而如果在传递过程中进行正规化，基本迭代次数在 1000 左右就能得到很好的效果。

2) 画出 loss-迭代次数曲线可以很好的找到最佳迭代次数，也可直接设置一个指定的误差常量，在 Loss 值不小于该常数时进行迭代。

十、代码及程序说明

本次提交代码有两份.py 文件分别为 NN.py,NNnew.py 文件。NN.py 为初始版本，能实现单隐藏层网络。NNnew.py 为优化后的版本，对代码结构进行了一定优化 (设置全局变量、对代码根据逻辑进行重排等)。如要进行测试请使用 NNnew.py 文件。该文件有全局变量 learningRate，iterations，它们可以设置学习速率与迭代次数。文件上方有文件路径，可直接修改以对不同数据集进行测试。测试者也可以通过修改网络结构测试不同的网络，只需要修改”Net() “括号中的数据。

程序还会动态显示数据集训练过程，将数据训练可视化。程序运行过程中输出的图片意义在结果说明中已经介绍就不再赘述。

参考文献

[1] “Learning representations by back-propagating errors,” *Nature*.