本周我主要学习了mlp算法，并找到了一些神经网络的程序运行并理解

Mlp算法我阅读了论文Remote Sensing Image Classification Model Based on Artificial Neural Network and Multi-layer Sensor\_MLP\_

多层感知器（MLP）是一类的[前馈](https://en.wikipedia.org/wiki/Feedforward_neural_network) [神经网络](https://en.wikipedia.org/wiki/Artificial_neural_network)。MLP由至少三层节点组成。除了输入节点，每个节点都是使用非线性[激活函数](https://en.wikipedia.org/wiki/Activation_function)的神经元。MLP利用一种被称为[反向传播](https://en.wikipedia.org/wiki/Backpropagation" \o "Backpropagation)来进行训练的[监督学习](https://en.wikipedia.org/wiki/Supervised_learning)技术。[[1]](https://en.wikipedia.org/wiki/Multilayer_perceptron#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Multilayer_perceptron#cite_note-2)其多层和非线性激活将MLP与线性[感知器](https://en.wikipedia.org/wiki/Perceptron" \o "Perceptron)区分开。它可以区分不可[线性分离的](https://en.wikipedia.org/wiki/Linear_separability)数据。

感知器是1957年美国学者Rosenblatt提出的一种用于模式分类的神经网络模型。当时的感知器模型只包括单结点的一个层,故称为单层感知器,它只适用于线性可分数据。而对于非线性数据,判别边界会产生振荡。为解决非线性可分数据的多类别分解问题,Rumelhart等人提出了多层感知器(Mul-tiple layer perceptron,MLP)模型。这种网络模型的结构是由不同层次的结点集合组成,每一层的结点通过与下一层互连输出到下一结点层。其输出层通过连接数值而被放大,衰减或抑制。除了输入层,每一结点的激励输出值由结点输入,激励函数及偏置量所决定。训练后的网络,可形成模式空间与分类空间的非线性映射关系。目前大多数基于人工神经网络(ANN)的遥感器处理与分析模型大多基于多层感知器。

1. MLP网络结构

MLP网络由感受层(S),联想层(A),响应层(R)构成。S,A,R均由同类神经元构成。

感受层为网络结构的输入层,用于特征向量的输入,

联想层为网络中隐含层,

而响应层则为网络的输出层。

S层单元与A层单元通过联结关系构成处理对象的联想矩阵,A层单元与R层单元之间的联结构成对处理对象的决策矩阵,通过训练整理,使网络形成有序的,具有决策能力的稳定结构。MLP网络对于多类别非线性可分数据的分类很有效。它包括3个或3个以上的层,输入层接受特征向量(Xn)的输入,在遥感影像分类中,该向量可为像元多波段数据向量、一定大小的影像结构窗口数据向量、其他复杂参数集构成的输入向量;隐层用来表示知识,采用分布式存储方法。一个包含足够多结点的隐层能表示所有非线性逻辑判别功能。增加隐层的结点数据能使网络具有更复杂的判别能力,但也降低了网络的综合能力和网络的训练时间。因此,选择合适的隐层结点数也是十分重要的。输出层对输入层信息的判别或决策结果进行输出,在遥感影像分类中,输出层的结点数等于分类的类别。MLP网络中每一结点都与前一层或后一层相互联,其他结点值通过相互间的联结权重确定。

2. MLP模型中采用的算法

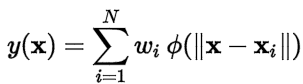
反向传播(Back Propagation,BP)学习算法在MLP模型中被最广泛应用。它是把一组样本的输入、输出问题变为一个非线性优化问题。网络开始训练时,选用较小的随机互联权值与内部阈值,通过反复加载训练样本并调整权值,直到代价函数下降到可接受的容限值。BP算法的学习过程,由正向传播和反向传播组成。在正向传播过程中,输入信息从输入层经过隐层,再传向输出层,每一层神经元的状态值只影响下一层神经元的状态值;如果在输出层不能得到期望的输出值,则转入反向传播,将误差信号沿逆向通路返回,通过修正各层神经元的权值,使得网络的总误差收敛到极小。

3激活函数

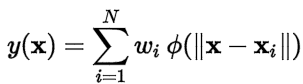
如果多层感知器在所有神经元中具有线性[激活函数](https://en.wikipedia.org/wiki/Activation_function" \o "激活功能)，即将[加权输入](https://en.wikipedia.org/wiki/Synaptic_weight)映射到每个神经元的输出的线性函数，则[线性代数](https://en.wikipedia.org/wiki/Linear_algebra)表明可以将任意数量的层减少到两层输入 -输出模型。在MLP中，一些神经元使用非线性激活函数，其被开发来模拟生物神经元的[动作电位](https://en.wikipedia.org/wiki/Action_potentials)或发射的频率。两个常见的激活函数都是[sigmoid](https://en.wikipedia.org/wiki/Sigmoids)，并且被描述第一个是[双曲正切](https://en.wikipedia.org/wiki/Hyperbolic_tangent)，范围从-1到1，而另一个是[逻辑函数](https://en.wikipedia.org/wiki/Logistic_function" \o "物流功能)，其形状类似，但范围从0到1.这里{\ displaystyle y\_ {i}}yi 是输出的{\ displaystyle i}ith节点（神经元）和 {\ displaystyle v\_ {i}}vi是输入连接的加权和。已经提出了另外的激活功能，包括[整流器和softplus](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)" \o "整流器（神经网络）)功能。更专业的激活功能包括[径向基函数](https://en.wikipedia.org/wiki/Radial_basis_functions" \o "径向基函数)（在[径向基网络中使用，](https://en.wikipedia.org/wiki/Radial_basis_network" \o "径向基网)另一类监督神经网络模型）。

径向基函数

径向基函数是一个取值仅仅依赖于离原点距离的实值函数，也就是Φ（x）=Φ(‖x‖),或者还可以是到任意一点c的距离，c点称为中心点，也就是Φ（x，c）=Φ(‖x-c‖)。任意一个满足Φ（x）=Φ(‖x‖)特性的函数Φ都叫做径向基函数，标准的一般使用欧氏距离（也叫做欧式径向基函数），尽管其他距离函数也是可以的。在神经网络结构中，可以作为全连接层和ReLU层的主要函数。

径向基函数通常用于建立[函数近似](https://en.wikipedia.org/wiki/Function_approximation" \o "函数近似)其中近似函数y（x）被表示为N个径向基函数的和，每个N个基函数与不同的中心x i相关联，并由适当的系数w i加权。权重瓦特我可使用的基体的方法来估计[线性最小二乘](https://en.wikipedia.org/wiki/Weighted_least_squares)，因为近似函数是线性的权重。

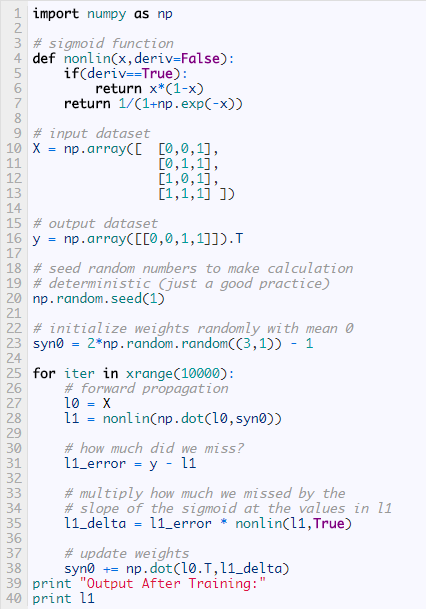
RBF神经网络

也可以解释为一个相当简单的单层类型的[人造神经网络，](https://en.wikipedia.org/wiki/Artificial_neural_network" \o "人工神经网络)称为[径向基函数网络](https://en.wikipedia.org/wiki/Radial_basis_function_network)，其中径向基函数起到网络激活功能的作用。如果使用足够大的数量*N*的径向基函数，原理上可以通过这种形式的和以任意精度来内插[紧凑](https://en.wikipedia.org/wiki/Compact_space" \o "紧凑的空间)间隔上的任何连续函数。

近似*y*（***x***）可以相对于权重*w i*是可微分的。因此，可以使用神经网络的任何标准迭代方法来学习权重。

以这种方式使用径向基函数产生合理的插值方法，条件是已经选择了拟合集，以便系统地覆盖整个范围（等距数据点是理想的）。然而，没有与径向基函数正交的多项式项，拟合集合以外的估计往往表现不佳。

接下来我找了一段简短的Python代码实现多层神经网络（一个用 BP 算法训练的神经网络尝试着用输入去预测输出。）



给定三列输入，试着去预测对应的一列输出。可以通过**测量输入与输出值的数据**来解决这一问题。直观意义上来讲，反向传播算法便是通过这种方式来衡量数据间统计关系进而得到模型的。

第一次运行之后提示NameError: name 'xrange' is not defined

后来经过百度发现是Python版本的问题，我的[**python**](http://lib.csdn.net/base/python)版本为python 3.6，而xrange( )函数时在python 2.x中的一个函数，在Python 3中，range()的实现方式与xrange()函数相同所以把xrange()换成range()错误就消失了

第二次运行之后发现没有报错但是无法显示预测结果，没有找到问题所在，于是我开始尝试理解代码

|  |  |
| --- | --- |
| X | 代表输入数据集，形式为矩阵，每 1 行代表 1 个训练样本。 |
| y | 代表输出数据集，形式为矩阵，每 1 行代表 1 个训练样本。 |
| l0 | 是网络第 1 层，即网络输入层。 |
| l1 | 是网络第 2 层，常称作隐藏层。 |
| syn0 | 是第一层权值，突触 0 ，连接 l0 层与 l1 层。 |
|  |  |
| x.dot(y) | 若 x 和 y 为向量，则进行点积操作；若均为矩阵，则进行矩阵相乘操作；若其中之一为矩阵，则进行向量与矩阵相乘操作。 |

**第 1 行：**这里导入一个名叫 numpy 的线性代数工具库

**第 4 行：**这里是“非线性”部分。虽然它可以是许多种函数，但在这里，使用的非线性映射为一个称作 “sigmoid” 的函数。Sigmoid 函数可以将任何值都映射到一个位于 0 到  1 范围内的值。通过它，我们可以将实数转化为概率值。

**第 5 行:** 通过 “nonlin” 函数体还能得到 sigmod 函数的导数

**第 10 行：**这行代码将我们的输入数据集初始化为 numpy 中的矩阵。每一行为一个“训练实例”，每一列的对应着一个输入节点。这样，神经网络便有 3 个输入节点，4 个训练实例。

**第  16 行：**这行代码对输出数据集进行初始化。以水平格式（ 1 行 4 列）定义生成了数据集。“.T” 为转置函数。经转置后，该  y  矩阵便包含 4 行 1 列。同我们的输入一致，每一行是一个训练实例，而每一列（仅有一列）对应一个输出节点。因此，网络含有 3 个输入， 1 个输出。

代码我会继续调试并尝试理解，并在下周尝试引入mlp算法。