机器学习应用实践（实验七）——MLP 网络与 BP 算法

一、 实验目的

1、掌握 Scikit-Learn 中提供的 MLP 网络算法的使用。

2、掌握 MLP 网络和 BP 算法的原理，并具备使用 python 实现 BP 神经网络的编程能力。

二、 实验准备

1、回顾 MLP 神经网络算法和 BP 算法的原理。

2、学习 Scikit-Learn 中 MLP 神经网络算法涉及到的相关知识。

三、 相关知识介绍

1、人工神经网络(ANN，Artificial Neural Network)模型

（1）神经元模型

神经元模型如图 1 所示，每个神经元都接受来自其他神经元的输入信号，每个信号都通过一个带有权重的连接传递，神经元把这些信号加起来得到一个总输入值，然后将总输入值与神经元的阈值进行对比，然后通过一个“激活函数”处理得到最终的输出 $y=f(z-\theta)=f(\sum_{i=1}^{n}w_i x_i-\theta)$，这个输出又会作为之后的神经元的输入一层层传递下去。其中“激活函数”是非线性变换，其作用是给神经元引入非线性因素，使神经网络可逼近任意非线性函数。

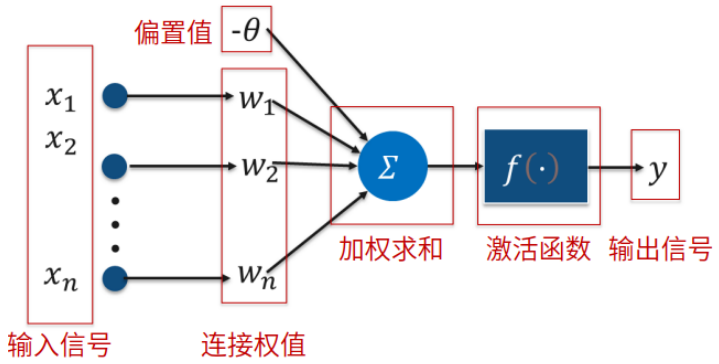


图 1 神经元模型

（2）多层前馈神经网络结构及符号定义

多层前馈神经网络的基本结构是第一层为输入层(Input Layer)，输入层只有一层，其中每个神经元为一个特征向量。最后一层为输出层(Output Layer)，输出层也只有一层，输出模型预测的结果。中间为隐层(Hidden Layer)，隐层至少一层，其中的每个神经元都是一个激活函数。因此多层前馈神经网络最少有 3 层，其结构如图 2 所示。多层感知机(Multi-Layer Perception,MLP)网络就是多层全连接前馈神经网络（全连接是指上一层的任何一个神经元与下一层的所有神经元都有连接）。

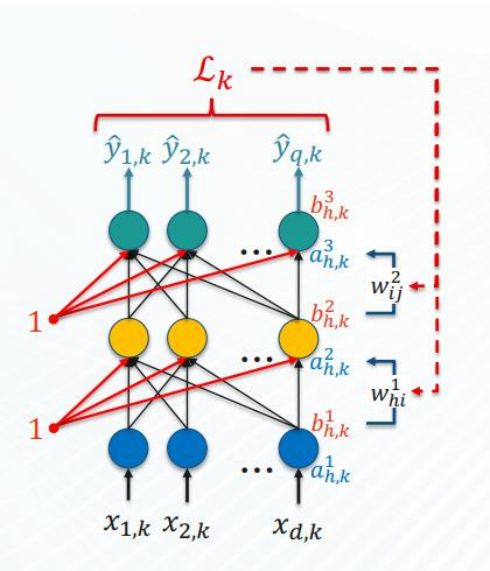


图 2 多层前馈神经网络基本结构

多层前馈神经网络中的符号定义如下：

- $D_{train} = \{(\boldsymbol{x}_k \in \Re^d, \boldsymbol{y}_k \in \Re^q) | k = 1,2, \dots N\}$
- $l = 1,2,3$，层的序号
- $h = 1,2, \dots, d$，第一层神经元序号
- $i = 1,2, \dots, p$，第二层神经元序号
- $j = 1,2, \dots, q$，第三层神经元序号
- $w^l_{hi}, l = 1$，第一层权重
- $w^l_{ij}, l = 2$，第二层权重
- $w^l_{hi}, l = 1, h = 0$，第一层偏置项
- $w^l_{ij}, l = 2, i = 0$，第二层偏置项
- 网络输出记为$\hat{y}_{j,k}, j = 1, \dots, q,\ k = 1, \dots, N$
- a：神经元输入(信号在进入激活函数前的加权累加值)
- b：神经元输出
- 对于第k个样本$\boldsymbol{x}_k = [x_{1,k}, \dots, x_{h,k}, \dots, x_{d,k}]$
  - $a^{l=1}_{h,k}$，第 1 层，第h个神经元的输入
  - $b^{l=1}_{h,k}$，第 1 层，第h个神经元的输出
  - $a^{l=2}_{i,k}$，第 2 层，第i个神经元的输入
  - $b^{l=2}_{i,k}$，第 2 层，第i个神经元的输出
  - $a^{l=3}_{j,k}$，第 3 层，第j个神经元的输入
  - $b^{l=3}_{j,k}$，第 3 层，第j个神经元的输出

（3）多层前馈神经网络的信号传递与计算

- 第一层

- 输入：$a_{h,k}^1 = x_{h,k}$
- 输出：$b_{h,k}^1 = a_{h,k}^1$

- 第二层
  - 输入：$a_{i,k}^2 = \sum_{h=0}^{d} w_{hi}^1 b_{h,k}^1$，$b_{h=0,k}^1 = 1$
  - 输出：$b_{i,k}^2 = S(a_{i,k}^2)$
  - 矢量表达：$\boldsymbol{b}_k^2 = S(W^1 \boldsymbol{b}_k^1)$
- 第三层
  - 输入：$a_{j,k}^3 = \sum_{i=0}^{p} w_{ij}^2 b_{i,k}^2$，$b_{i=0,k}^2 = 1$
  - 输出：$b_{j,k}^3 = S(a_{j,k}^3)$
  - 矢量表达：$\boldsymbol{b}_k^3 = S(W^2 \boldsymbol{b}_k^2)$

2、BP 算法原理

由于多层前馈神经网络的训练经常采用误差反向传播(Back Propagation)算法，人们也常把采用 BP 算法的多层前馈神经网络称为 BP 神经网络(Back Propagation Neural Network)。BP 算法的学习过程是由信号的正向传播与误差的反向传播两个过程组成。

正向传播时，输入样本从输入层进入网络，经隐层逐层传递至输出层，如果输出层的实际输出与期望输出不同，则转至误差反向传播；如果输出层的实际输出与期望输出相同，结束学习算法。

反向传播时，将输出误差(期望输出与实际输出之差)按原通路反传计算，通过隐层反向，直至输入层，在反传过程中将误差分摊给各层的各个单元，获得各层各单元的误差信号，并将其作为修正各单元权值的根据。这一计算过程使用梯度下降法完成，在不停地调整各层神经元的权值和阈值后，使误差信号减小到最低限度。

权值和阈值不断调整的过程，就是网络的学习与训练过程，经过信号正向传播与误差反向传播，权值和阈值的调整反复进行，一直进行到预先设定的学习训练次数，或输出误差减小到允许的程度。

算法步骤如下：

输入	训练数据集：$\{(x_k, y_k) \mid k = 1,2,\dots,N\}$
1	网络初始化设置：
	a) 设定MLP网络层数L，每一层神经元个数$n_l, l = 1,\dots,L$； b) 设定每一层神经元激活函数$S_l(a)$； c) 设定网络损失函数$\mathcal{L}$； d) 设定学习率η和优化算法； e) 设定终止准则(精度、最大迭代次数等)； f) 网络权重初始化(小随机数)；
2	输入训练样本，前向传播，计算神经元输入a,输出b，最终输出$\hat{y}$；
3	根据真实标签y与当前预测$\hat{y}$，计算各层神经元对应损失梯度g；
4	利用梯度下降法，对各层连接权的权重进行更新；
5	迭代输入训练样本，重复步骤2~4，直到满足终止条件。
输出	训练收敛的MLP网络模型

（1）反向传播算法推导

最小化神经网络的损失函数：$W^* = \underset{W}{\operatorname{argmin}}\,\mathcal{L}(W;D)$

对于回归任务，损失函数为平方误差为：

$$\mathcal{L}(W;D) = \frac{1}{2}\sum_{k=1}^{N}\|\hat{\boldsymbol{y}}_{\boldsymbol{k}} - \boldsymbol{y}_{\boldsymbol{k}}\|^2 = \frac{1}{2}\sum_{k=1}^{N}\|f(\boldsymbol{x}_k;W) - \boldsymbol{y}_{\boldsymbol{k}}\|^2$$

对于分类(二分类)任务，损失函数为交叉熵损失为：

$$\mathcal{L}(W;D) = -\frac{1}{N}\sum_{k=1}^{N}[\boldsymbol{y}_k^T\, ln\,\hat{\boldsymbol{y}}_k + (1-\boldsymbol{y}_k^T)\, ln(1-\hat{\boldsymbol{y}}_k)]$$

$$= -\frac{1}{N}\sum_{k=1}^{N}\left[\boldsymbol{y}_k^T\, ln\, f(\boldsymbol{x}_k;W) + (1-\boldsymbol{y}_k^T)\, ln\big(1-f(\boldsymbol{x}_k;W)\big)\right]$$

- 对于单个样本$(\boldsymbol{x}_k, \boldsymbol{y}_k)$
  - 目标函数记为$\mathcal{L}_k$
  - 待优化变量$W = \{w_{hi}^1, w_{ij}^2 | h,i,j\}$
- 求解策略
  - 梯度下降法：$\partial\mathcal{L}_k/\partial W$
  - 基于链式法则的梯度传递
    - 目标函数$\boldsymbol{\mathcal{L}_k}$对输出层权重$w_{ij}^2$求导
    - $\frac{\partial\mathcal{L}_k}{\partial w_{ij}^2} = \frac{\partial\mathcal{L}_k}{\partial\hat{y}_{j,k}}\frac{\partial\hat{y}_{j,k}}{\partial a_{j,k}^3}\frac{\partial a_{j,k}^3}{\partial w_{ij}^2}$

其中：

- $\frac{\partial\mathcal{L}_k}{\partial\hat{y}_{j,k}} = \mathcal{L}_k'(\hat{y}_{j,k})$
- 对于回归任务，$\mathcal{L}_k'(\hat{y}_{j,k}) = \hat{y}_{j,k} - y_{j,k}$
- 对于分类任务，$\mathcal{L}_k'(\hat{y}_{j,k}) = \frac{1-y_{j,k}}{1-\hat{y}_{j,k}} - \frac{y_{j,k}}{\hat{y}_{j,k}}$
- $\frac{\partial\hat{y}_{j,k}}{\partial a_{j,k}^3} = \frac{\partial b_{j,k}^3}{\partial a_{j,k}^3} = \frac{\partial S(a_{j,k}^3)}{\partial a_{j,k}^3} = S'(a_{j,k}^3)$
- $\frac{\partial a_{j,k}^3}{\partial w_{ij}^2}$

  - 目标函数$\mathcal{L}_{\mathrm{k}}$对输出层权重$\mathrm{w}_{\mathrm{hi}}^1$求导
  - $\frac{\partial\mathcal{L}_k}{\partial w_{hi}^1} = \sum_{j=1}^{q}\left[\frac{\partial\mathcal{L}_k}{\partial\hat{y}_{j,k}}\frac{\partial\hat{y}_{j,k}}{\partial a_{j,k}^3}\frac{\partial a_{j,k}^3}{\partial b_{i,k}^2}\right]\frac{\partial b_{i,k}^2}{\partial a_{i,k}^2}\frac{\partial a_{i,k}^2}{\partial w_{hi}^1}$

其中：

- $\frac{\partial \mathcal{L}_k}{\partial \hat{y}_{j,k}} \frac{\partial \hat{y}_{j,k}}{\partial a^3_{j,k}} = g_{j,k}$
- $\frac{\partial a^3_{j,k}}{\partial b^2_{i,k}} = \frac{\partial \sum_{i=0}^{p} w^2_{ij} b^2_{i,k}}{\partial b^2_{i,k}} = w^2_{ij}$
- $\frac{\partial b^2_{i,k}}{\partial a^2_{i,k}} = b^2_{i,k}\left(1 - b^2_{i,k}\right)$
- $\frac{\partial a^2_{i,k}}{\partial w^1_{hi}} = \frac{\partial \sum_{i=0}^{p} \partial w^1_{hi} b^1_{h,k}}{\partial w^1_{hi}} = b^1_{h,k}$

（2）更新策略——梯度下降法

- 隐层到输出层权重w^2_{ij}的更新
  - $w^2_{ij} = w^2_{ij} - \eta_2 \Delta w^2_{ij} = w^2_{ij} - \eta_2 g_{j,k} b^2_{i,k}$
- 输入层到隐层权重w^1_{hi}的更新
  - $w^1_{hi} = w^1_{hi} - \eta_1 \Delta w^1_{hi} = w^1_{hi} - \eta_1 g_{i,k} b^1_{h,k}$

3、sklearn 中 BP 算法 MLPClassifier API 介绍 （MLP 又称多层感知机）

sklearn 是专注于机器学习的库，它在神经网络的模块中特地标注：sklearn 不是用于深度学习的平台，因此这个神经网络不具备做深度学习的功能，也不具备处理大型数据的能力。sklearn 中提供了如表 1 所示的神经网络类供用户使用。其中两个以多层感知机为基础的类分别为 MLPClassifier 和 MLPRegressor。两个类的参数比较类似，本文以多层感知机分类器 MLPClassifier 为例介绍其中的重要参数、属性和方法的含义和使用。

表 1 sklearn 中提供的神经网络类

类	含义
neural_network.BernoulliRBM	伯努利限制玻尔兹曼机器(RBM)，这是一种随机递归神经网络。
neural_network.MLPClassifier	多层感知机分类器
neural_network.MLPRegressor	多层感知机回归器

与其他分类器一样，可以通过“实例化、fit、predict/score”三部曲来使用 MLPClassifier。程序示例如下：

```
>>> from sklearn.neural_network import MLPClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
...                                                     random_state=1)
>>> clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
>>> clf.predict_proba(X_test[:1])
array([[0.038..., 0.961...]])
>>> clf.predict(X_test[:5, :])
array([1, 0, 1, 0, 1])
>>> clf.score(X_test, y_test)
0.8...
```

（1）重要参数说明

- hidden_layer_sizes：字典类型，字典中的第 i 个元素代表第 i 个隐层中的神经元数量。默认为(100,)，即只有一层隐层，神经元个数为 100。若 hidden_layer_sizes=(50, 50)，表示有两层隐层，第一层隐层有 50 个神经元，第二层隐层也有 50 个神经元。
- activation：激活函数，包括：‘identity’, ‘logistic’, ‘tanh’, ‘relu’， 默认使用‘relu’。
  1) identity：f(x) = x
  2) logistic：其实就是 sigmod，f(x) = 1 / (1 + exp(-x))
  3) tanh：f(x) = tanh(x)
  4) relu：f(x) = max(0, x)
- solver：求解器，提供三种求解器：‘lbfgs’, ‘sgd’, ‘adam’， 默认‘adam’来优化权重。
  1) lbfgs：quasi-Newton 方法的优化器
  2) sgd：随机梯度下降
  3) adam： Kingma, Diederik, and Jimmy Ba 提出的基于随机梯度的优化器

注意：默认求解器 ‘adam’在相对较大的数据集上效果比较好（几千个样本或者更多），对小数据集来说，‘lbfgs’收敛更快效果也更好。

- alpha：float，L2 正则化项的强度(L2 正则化项在添加到损失时除以样本大小)。默认 0.0001。
- batch_size：int，随机优化器的小批量大小。如果求解器为“lbfgs”，则分类器将不使用 minibatch。当设置为“auto”时，batch_size=min（200，n_samples）。
- learning_rate：用于权重更新的学习率，只有当 solver 为’sgd’时使用。可选项包括{‘constant’，’invscaling’, ‘adaptive’}，默认 constant 。
  1) ‘constant’：有’learning_rate_init’给定的恒定学习率 。
  2) ‘incscaling’：随着时间 t 使用’power_t’的逆标度指数不断降低学习率 learning_rate_ ，effective_learning_rate = learning_rate_init / pow(t, power_t) 。
  3) ‘adaptive’：只要训练损耗在下降，就保持学习率为’learning_rate_init’不变，当连续两次不能降低训练损耗或验证分数停止升高至少 tol 时，将当前学习率除以 5。
- learning_rate_int：float，使用的初始学习率，它控制更新权重时的步长。仅在求解器为’sgd’或‘adam’时使用，默认 0.001。
- power_t：float，是逆比例学习率的指数。只有 solver=’sgd’时使用。默认为 0.5。当 learning_rate=’invscaling’，用来更新有效学习率。
- max_iter：int，最大迭代次数，默认 200。
- random_state：int 或 RandomState，可选，默认 None，随机数生成器的状态或种子。 决定用于权重和偏差初始化的随机数生成。
- shuffle：bool，判断是否在每次迭代时对样本打乱顺序，默认为 True，只有当 solver=’sgd’或者‘adam’时使用。
- tol：float, 优化迭代的阈值，默认 1e-4。

- verbose：bool，可选，默认 False，是否将过程打印到 stdout。
- warm_start：bool，可选，默认 False，当设置成 True，使用之前的解决方法作为初始拟合，否则释放之前的解决方法。
- momentum：float，动量梯度下降更新，设置的范围应该在 0-1 之前。只有 solver='sgd'时使用。默认 0.9。
- nesterovs_momentum：bool，是否使用动量法进行迭代。默认为 True。只有 solver='sgd'并且 momentum > 0 使用。
- early_stopping：bool，默认 False。当验证分数没有提高时，是否使用提前停止来终止训练。如果设置为 true，将自动留出 10%的训练数据作为验证集，当验证集上的损失函数值连续 n_iter_no_change 次没有下降或下降量不达阈值 tol 时，则终止训练。如果提前停止为 False，则当训练集上连续 n_iter_no_change 次没有下降或下降量不达阈值 tol 时，停止训练。仅当 solver='sgd'或'adam'时有效。
- validation_fraction：float，默认 0.1，用作早期停止验证的预留训练数据集的比例，取值在 0-1 之间，只当 early_stopping=True 有用 。
- beta_1：float，默认 0.9，只有 solver='adam'时使用，估计一阶矩向量的指数衰减速率，[0,1)之间。
- beta_2：float，默认 0.999，只有 solver='adam'时使用，估计二阶矩向量的指数衰减速率[0,1)之间。
- epsilon：float，默认 1e-8，只有 solver='adam'时使用数值稳定值。

（2）属性说明

属性	说明
classes_	ndarray or list of ndarray of shape (n_classes,) 每个输出的类别标签。
loss_	float, 用损失函数计算的当前损失。
coefs_	list, length n_layers – 1 列表中的第 i 个元素表示与第 i 层相对应的权重矩阵。
intercepts_	list, length n_layers – 1 列表中的第 i 个元素表示与层 i+1 对应的偏差向量。
n_iter_	int, 求解程序已运行的迭代次数。
n_layers_	int, 层数。
n_outputs_	int, 输出数量。

（3）方法说明

方法	说明
fit（X，y）	根据矩阵 X 和标签 y 的数据拟合模型
get_params([deep])	获取这个估计器的参数。
predict(X)	使用多层感知器分类器进行预测

方法	说明
predict_log_proba(X)	返回概率估计的对数。
predict_proba(X)	概率估计。
score(X, y[, sample_weight])	返回给定测试数据和标签上的平均准确率。
set_params(**params)	设置这个估计器的参数。

四、 实验内容

1、题目一（选做）：采用 scikit-learn 中的 MLPClassifier 对红酒数据集进行分类，并通过特征和边界的可视化，直观体会多层感知机网络中的隐层上神经元数量、隐层层数、激活函数、正则化项系数等超参数对模型复杂程度的影响。

具体内容：

（1）选取前两个特征，建立多层感知机网络进行多分类。

（2）可视化：通过散点图可视化数据样本（之前选择的两个特征），并画出模型训练后得到的决策边界。

【讨论一】改变单隐层中神经元个数（如 10 个，100 个），其他参数不变，观察其对决策边界的影响。

【讨论二】改变神经网络深度（如深度为 2，每层 10 个神经元），其他参数不变，与讨论一进行对比，观察神经网络深度其对决策边界的影响。

【讨论三】在讨论一（或讨论二）的基础上，改变激活函数（如 tanh、relu），与讨论一（或讨论二）进行对比，观察不同激活函数对决策边界的影响。

【讨论四】在讨论三的基础上，增大正则化系数，观察正则化对决策边界的影响。

总结：综合上述讨论，隐层上神经元数量、隐层层数、激活函数、正则化项系数对模型复杂程度有何影响。

2、题目二：采用 scikit-learn 中的 MLPClassifier 对自带手写数字数据集进行分类。

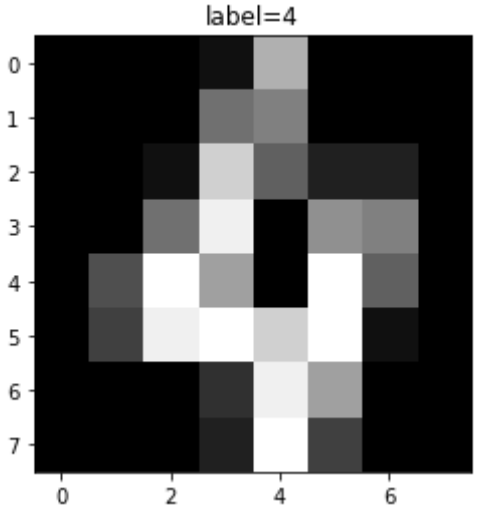
具体要求：

（1）导入数据集：手写数字集是 sklearn 中自带的数据集，它是一个三维数组(1797, 8, 8)， 即有 1797 个手写数字，每个数字由 8*8 的像素矩阵组成。矩阵中每个元素都是 0-16 范围内的整数。分类标签为 0-9 的数字。程序示例：

```
from sklearn.datasets import load_digits
digits_X,digits_y = load_digits(return_X_y=True)
```

数组形式数据可通过下面代码查看每个样本对应的手写数字图像。

```
# 显示手写数字集中的一个样本图像
plt.imshow(digits_X[4].reshape(8,8),cmap="gray")
plt.title("label="+str(digits_y[4]))
```



（2）模型建立：使用 MLPClassifier 建立分类模型。

（3）输出：分类结果的准确率。

【讨论五】结合模型复杂度与模型泛化误差之间的关系，调节模型超参数，提升模型泛化性能。可尝试调节隐层神经元个数和隐层数、激活函数、学习率、正则项系数等超参数。

3、题目三：编写 BPNN 算法，对 iris 数据集/手写数字集进行二分类或多分类。

具体要求：

（1）数据样本标签处理：

二分类任务：正类为 1，负类为 0。

多分类任务：将样本标签变为 one-hot 向量。

（2）搭建浅层神经网络：隐层数 1-2 个即可。每层神经元的个数自选。

（3）激活函数：自选（relu、sigmoid、tanh）。

代价函数：自选（交叉熵损失、均方误差）。

（4）输出：分类准确率。

（5）可视化：迭代的代价函数曲线。

（6）手写 BP 网络链式法则的反向传播计算过程，拍照附在报告上。

注意：

（1）为提高运算效率，算法编写尽量使用向量化技术，避免使用 for 循环遍历样本和神经元。

（2）一般将权重 w 和偏置 b 分开进行计算。

（3）为避免梯度消失或梯度爆炸，通常采用随机初始化权重 w，而不是将权重全部初始化为 0 或 1。同时，这也是除归一化处理之外，另一个在训练深度网络时，加快训练速度的方法。

示例：若要生成服从 $\mathcal{N}(0,\sqrt{\frac{2}{n_{in}+n_{out}}})$分布的随机数，可使用如下程序：

np.random.randn(m,n) * $\sqrt{\frac{2}{n_{in}+n_{out}}}$，其中$n_{in}$为神经元的输入连接数量，$n_{out}$为神经元的输出连接数量，m 和 n 分别为返回数组的行数和列数。

（4）向量计算过程中，要注意数组维度，为避免出现异常 bug，数组创建不要使用一维数组(例如 np.random.randn(5))，这不是列向量也不是行向量！可通过下面示例 np.random.randn(5,1)或量 np.random.randn(1,5)创建列向量或行向量，或使用 reshape 方法变换数组的维度。