

BP实验报告

- 实验内容
- 实验模型
- 实验原理和公式参考
- 实验设计
  - 数据预处理
  - 数据集划分
  - 评价指标
  - 实验思路和代码框架
- 文件结构和代码运行说明
  - 文件结构
  - 运行说明
    - 平台说明
    - BP正确性验证
    - 分类性能测试

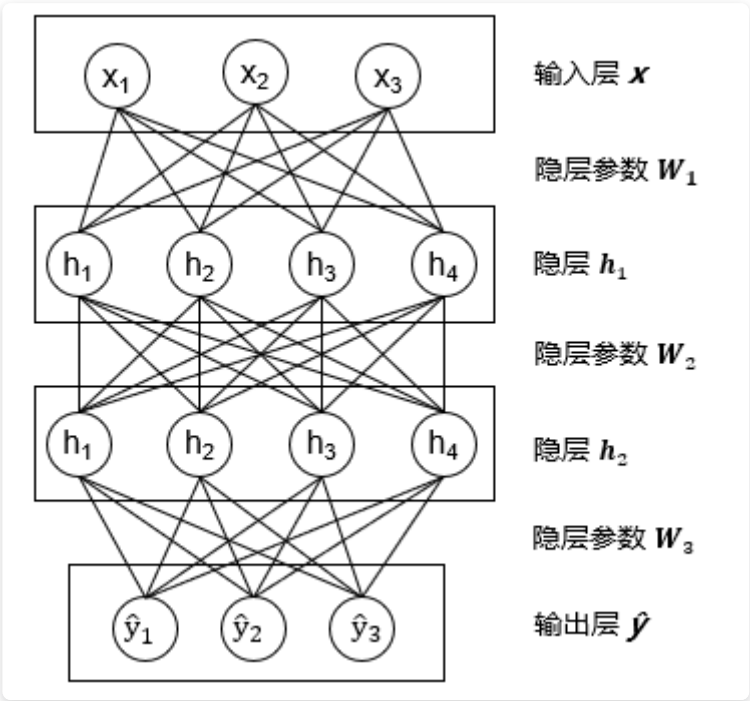
# BP实验报告

## 实验内容

- 实现一个四层的感知机模型
- 实现BP算法与梯度下降算法
- 实现简单的多分类任务

## 实验模型

实验采用BP算法多层感知机模型，其模型具体如下图所示。要求不含有偏置向量 $b_i$ 。



多层感知机模型

另一方面，为了提高系统的拓展性，实验设计中，对于隐含层 $h_1, h_2$ 的隐含层结点数都是可变的，甚至，对于隐含层的个数也可以改变。这部分将在后面进行说明

# 实验原理和公式参考

网络输入输出之间的公式表达:

$$\begin{aligned}h_1 &= s_1(W_1x) \\h_2 &= s_2(W_2h_1) \\\hat{y} &= s_3(W_3h_2) \\L &= \ell(y, \hat{y})\end{aligned}\tag{1}$$

求解的具体问题达标式:

$$\begin{aligned}\frac{\partial L}{\partial W_1} &= (W_2^T(W_3^T(\ell' s'_3) \odot s'_2) \odot s'_1)x^T \\\frac{\partial L}{\partial W_2} &= (W_3^T(\ell' s'_3) \odot s'_2)h_1^T \\\frac{\partial L}{\partial W_3} &= (\ell' s'_3)h_2^T\end{aligned}\tag{2}$$

其中有:

$$\begin{aligned}s_1 &= s_2 = \sigma \\\sigma' &= \sigma(1 - \sigma) \\s_3(x_1, x_2, x_3) &= \text{Softmax}(x_1, x_2, x_3) \\&= \frac{1}{e^{x_1} + e^{x_2} + e^{x_3}}(e^{x_1} + e^{x_2} + e^{x_3}) \\\ell(y, \hat{y}) &= \text{CrossEntropy}(y, \hat{y}) = -\log \hat{y}_i, i = y \\(\ell' s'_3)_i &= \begin{cases} \hat{y}_i - 1, i = y \\ \hat{y}_i, i \neq y \end{cases}\end{aligned}$$

梯度下降算法表达式为:

$$W_i = W_i - \eta \frac{\partial L}{\partial W_i}\tag{3}$$

## 实验设计

### 数据预处理

需要完成将非实型数据转化为实型数据的功能, 实验中采用onehot编码的方式, 代码见MyUtil文件中的 `load_data(path = "iris.data")->list` 函数, 其中onehot编码功能如下所示

```
# one_hot encoder
target_label = 'classes'
target_class = iris[target_label]
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(target_class)
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)###
```

### 数据集划分

数据集划分为不相交的训练集、验证集、测试集, 划分数据集前一般会打乱数据集以随机采样, 代码见MyUtil文件中的 `train_test_validate_split(datas, labels, ratio = [0.8, 0.1, 0.1], random_state = 0)->list` 函数, 其中打乱数据集并拆分的编码功能如下所示

```

ratio_train = ratio[0] #训练集比例
ratio_validate = ratio[1] #验证集比例
ratio_test = ratio[2] #测试集比例
assert (ratio_train + ratio_validate + ratio_test) == 1.0, 'Total ratio Not equal to 1' ##检查总比例是否等于1
cnt_train = int(len(Datas) * ratio_train)
cnt_test = int(len(Datas) * ratio_test)
cnt_validate = len(Datas) - cnt_train - cnt_test
train_x = Datas[0:cnt_train]
train_y = Labels[0:cnt_train]
validate_x = Datas[cnt_train:cnt_train + cnt_validate]
validate_y = Labels[cnt_train:cnt_train + cnt_validate]
test_x = Datas[cnt_train + cnt_validate:]
test_y = Labels[cnt_train + cnt_validate:]

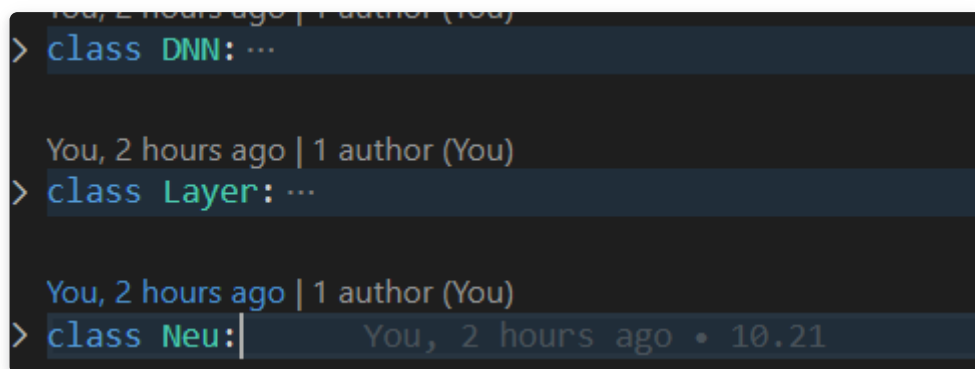
```

## 评价指标

实验中对准确率，精确率，召回率，F1值等都进行了评估，这里不做赘述

## 实验思路和代码框架

实验中设计了三个类，分别为DNN，Layer和Neu类，这里做一些简单说明。



```

> class DNN: ...

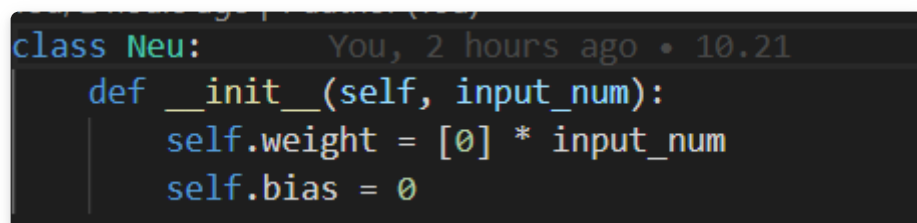
You, 2 hours ago | 1 author (You)
> class Layer: ...

You, 2 hours ago | 1 author (You)
> class Neu: | You, 2 hours ago • 10.21

```

实验中设计的三个类

- Neu类为每一个神经元的抽象，由于不考虑偏置向量，因此Neu类只有权重属性，是每一个输入到该神经元的权重。



```

class Neu: | You, 2 hours ago • 10.21
    def __init__(self, input_num):
        self.weight = [0] * input_num
        self.bias = 0

```

Neu类

- Layer类为每一层神经元的抽象，layer层包括以下几个函数，

```
class Layer:
    def __init__(self, input_num = 4, neu_num = 4, weight = None, activate_func = 'sigmoid'): ...
    def forward(self, input) -> np.matrix: ...
    def softmax(self, x) -> np.matrix: ...
    def sigmoid(self, input): ...
    def backPropS_i(self): ...
```

### Layer类中的函数

函数功能说明:

- `__init__`: 初始化, 此外会生成一个 `Neu_pools` 用于存储本层包含的 `Neu` 实例

参数说明

```
:param input_num: 输入的神经元数
:param neu_num: 本层的神经元数
:param weight: 本层的初始化权重
:param activate_func: 本层的激活函数
```

- `forward`: 前向传递

参数说明

```
:param input: 输入
:return 输出
```

- `softmax`: 手动实现的softmax, 不做赘述
- `sigmoid`: 手动实现的sigmoid, 不做赘述
- `backPropS_i`: 手动实现的sigmoid求导, 不做赘述
- DNN类为网络的抽象, 其中包含以下几个函数:

```
class DNN:
    def __init__(self, layer = [4,10,20,3], weight = None): ...
    def forward(self, input) -> np.matrix: ...
    def step(self) -> None: ...
    def cross_entropy(self, out, label) -> float: ...
    def backProp(self, output, label, lr = 0.06, skip = False) -> float: ...
    def backPropLoss(self, output, label) -> np.matrix: ...
    def show_grad(self): ...
    def train_validate(self, X, Y, Vx, Vy, Epochs = 300, batch = 12, lr = 0.05, show = True, grad_show = False) -> None: ...
    def test(self, X, Y, str) -> None: ...
```

### DNN类的函数

函数功能说明:

- `__init__`: 初始化, 此外会生成一个 `layer_pools` 用于存储网络包含的 `Layer` 实例

参数说明

:param layer: 从输入层到输出层, 每一层的结点个数, layer的长度对应于网络的深度, 每一个元素值对应该层的节点个数, 从而创建网络  
:param weight: 网络每一层的权重初始化数值

- forward:前向传递

参数说明

:param input:输入  
:return 网络的最终输出

- step:更新梯度计算权重
- cross\_entropy:手动实现的交叉熵函数, 不做赘述
- backProp:bp算法核心,计算的梯度将会存在每一层layer.partial属性中

:param output: 网络输出值  
:param label: 训练的目标值  
:param lr: 学习率  
:param skip: 验证集用于跳过权值更新的flag  
:return : 网络的误差

- backPropLoss:计算网络输出值和训练目标值之间的误差向量
- show\_grad: 用于梯度打印
- train\_validate:训练和验证核心代码

:param X: 用于训练的features  
:param Y: 用于训练的目标值  
:param vx: 用于验证的features  
:param vy: 用于验证的目标值  
:param Epoch: 迭代次数  
:param batch: 一次参与训练的样本数  
:param lr: 学习率  
:param show: 是否绘制loss的flag  
:param grad\_show: 是否打印梯度的flag

- test:测试核心代码, 包括准确率, 精确率, 召回率, F1值等指标

:param X: 用于测试的features  
:param Y: 用于测试的目标值  
:param strs: 提示string

## 文件结构和代码运行说明

### 文件结构

提交文档中包含以下几个文件:

- BP\_report.pdf: 实验报告的pdf版本
- main.py: 可运行文件, 使用是实验的BP算法等对iris数据集分析, 最终打印出性能指标, 以及图形绘制
- BPTTest.py: 可运行文件, 用于检验实现的BP算法正确性, 运行该文件即可

- iris.data,iris.name: 实验提供的数据, 不做过多说明
- MyUtil.py:包含以下两个函数, 用于数据集加载和数据集划分

```
def load_data(path = "./iris.data")->list:
def train_test_validate_split(datas, labels, ratio = [0.8,0.1,0.1],
random_state = 0)->list:
```

- DNN.py:自己实现的BP神经网络以及相关类
- DNN\_PYTORCH:利用pytorch搭建的神经网络
- requirements.yml: 依赖的库(实验中只涉及pandas,torch,numpy和matplotlib)

## 运行说明

### 平台说明

- 开发工具: VSCode 1.50.1
- OS: Windows\_NT x64 10.0.18363
- 编程语言: Python3.7.6

### BP正确性验证

运行说明, 运行BPTTest.py文件即可,下对该文件部分代码进行说明。

- 对实验数据中的第一个样本进行测试, 分别使用自己实现的BP网络和pytorch的自动求解梯度
- 网络权重随机赋值, 并且赋值给两种实现的网络进行初始化, 初始化数值如下产生, 由于采用相同的值对两个网络初始化, 因此随机种子是否固定不会影响功能的检查。

```
weight12 = np.random.normal(loc=0., scale=1., size=(layer[1], layer[0])) / np.sqrt(layer[0])
weight23 = np.random.normal(loc=0., scale=1., size=(layer[2], layer[1]))
weight34 = np.random.normal(loc=0., scale=1., size=(layer[3], layer[2]))
weight = [weight12,weight23,weight34]
```

#### 生成用于初始化的网络权重

- `layer = [4,4,4,3]`: 并非固定, 如示例则生成一个4-4-4-3网络, 两个隐含层结点数可以自由更改, 例如 `layer = [4,10,20,3]` 就会生成一个4-10-20-3网络。
- 运行该程序, 例如设定网络结构 `layer = [4,4,4,3]`, 即4-4-4-3网络, 在经过权重初始化后, 程序运行结果如下所示:

```

PS C:\Users\陈勇虎\Desktop\Project> & C:/ProgramData/Anaconda3/python.exe c:/Users/陈勇虎/Desktop/Project/Project1/BPTest.py
矩阵实现的BP算法
网络层梯度按照从输入层到输出层的顺序依次为:

[[ -0.00244  -0.001674 -0.00067  -0.000096]
 [ 0.055686  0.038216  0.015286  0.002184]
 [ -0.041503  -0.028483  -0.011393  -0.001628]
 [ -0.277826  -0.190665  -0.076266  -0.010895]]

[[0.000083  0.026019  0.0024  0.023665]
 [0.000089  0.027674  0.002552  0.025171]
 [0.000381  0.11882  0.010959  0.10807 ]
 [0.000364  0.113632  0.01048  0.103351]]

[[ -0.566866  -0.369622  -0.326541  -0.258107]
 [ 0.24931  0.162562  0.143615  0.113517]
 [ 0.317555  0.20706  0.182927  0.14459 ]]

pytorch自动求梯度
网络层梯度按照从输入层到输出层的顺序依次为:

[[ -0.00244  -0.001674 -0.00067  -0.000096]
 [ 0.055686  0.038216  0.015286  0.002184]
 [ -0.041503  -0.028483  -0.011393  -0.001628]
 [ -0.277827  -0.190665  -0.076266  -0.010895]]

[[0.000083  0.026019  0.0024  0.023664]
 [0.000089  0.027674  0.002552  0.025171]
 [0.000381  0.11882  0.010959  0.10807 ]
 [0.000364  0.113632  0.01048  0.103351]]

[[ -0.566866  -0.369622  -0.326541  -0.258107]
 [ 0.24931  0.162562  0.143615  0.113517]
 [ 0.317555  0.20706  0.182927  0.14459 ]]

```

### 随机初始化后的一组程序输出

首先依次输出自己实现的BP网络，在第一次BP过程中计算的三个梯度矩阵，随后一次输出用过pytorch自动求解梯度获得结果。

输出结果保留6位小数，可见两个的输出相同，说明实现的BP算法正确。

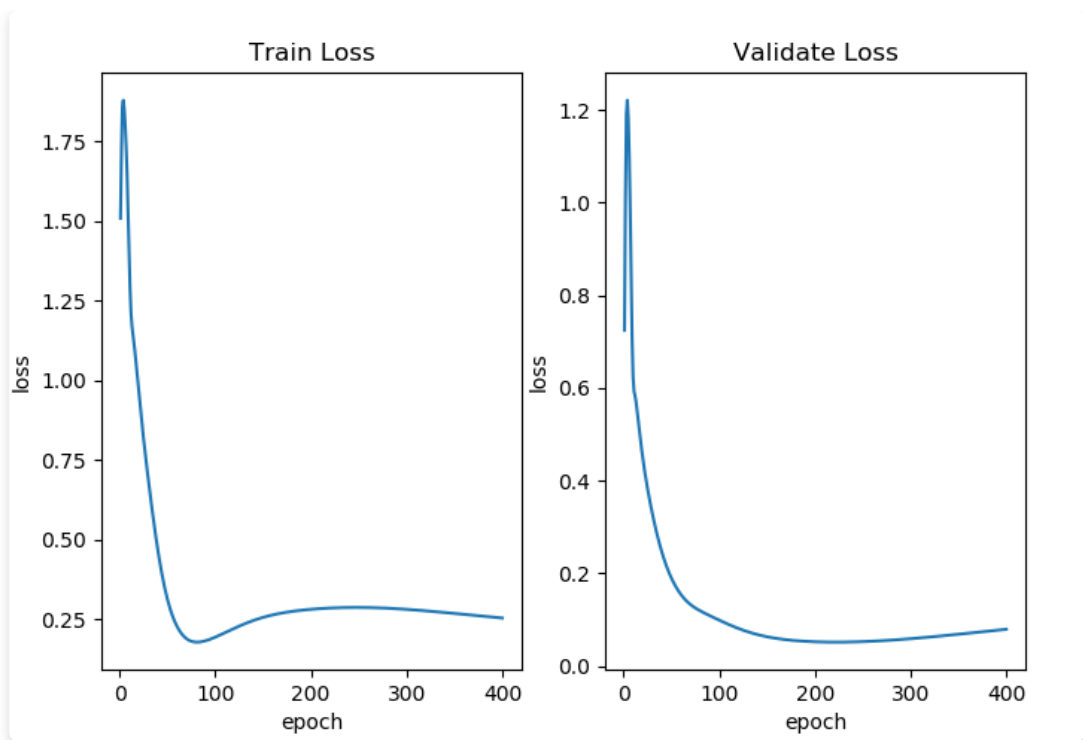
PS: 可以更改 `layer = [4,4,4,3]` 为 `layer = [4,x,y,3]`,  $x, y \in N^*$ , 进行更多组的测试。

## 分类性能测试

类似于BP正确验证的设置，layer层和权重初始化同上。

在实验中，设定 `epoch = 400`, `lr = 0.05`, `layer = [4,10,20,3]`, 数据集划分 `ratio = [0.8,0.1,0.1]`, `random_state = 0`, `batch_size = 12`, 以上参数可以在 `main.py` 的 `main()` 中自由修改。在如上的设置下，进行的一组测试输出依次如下：

- 训练集和验证集的Loss曲线：



训练集，验证集loss曲线

- 性能输出:性能将会输出到终端，在其中一组数据下测试结果为:

```
Train data evaluation
准确率 : 96.67%
精确率 : 94.44%
召回率 : 97.33%
F1值 : 95.87%
Validate_data evaluation
准确率 : 100.00%
精确率 : 100.00%
召回率 : 100.00%
F1值 : 100.00%
Test_data evaluation
准确率 : 100.00%
精确率 : 100.00%
召回率 : 100.00%
F1值 : 100.00%
准确率 : 100.00%
```

终端输出的性能

- 由于网络权重的初始化时随机的，因此每一次测试结果会略有不同，但是性能基本稳定。

写在最后:

文档中包含了对代码运行的屏幕录屏，由于开发IDE使用Visual Studio Code, 并没有使用Pycharm，若在运行结果方面与实验报告不一致，请与我联系。



