MLP

1. 实验目的

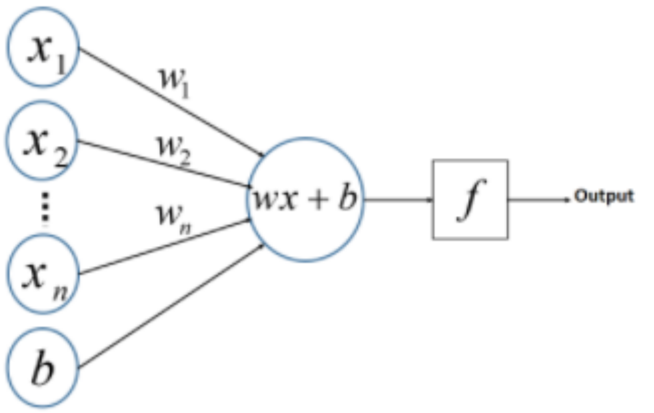
了解多层感知机算法的原理，并且可以简单应用

2. 算法原理

1)感知机

多层感知机是由感知机推广而来，感知机学习算法(PLA: Perceptron Learning Algorithm)用神经元的结构进行描述的话就是一个单独的。

感知机的神经网络表示如下：



$$u=\sum_{i=1}^{n} w_i x_i + b$$

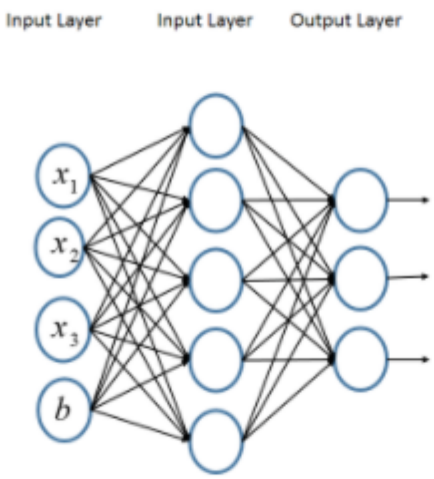
$$y = sign(u) = \begin{cases} +1, & u > 0 \\ -1, & u \leq 0 \end{cases}$$

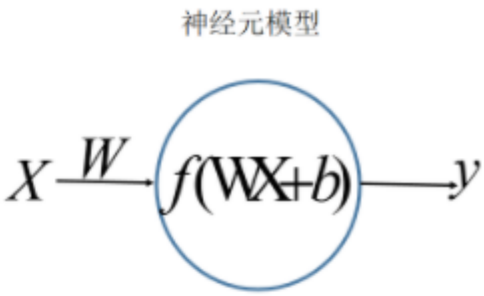
从上述内容更可以看出，PLA 是一个线性的二分类器，但不能对非线性的数据并不能进行有效的分类。因此便有了对网络层次的加深，理论上，多层网络可以模拟任何复杂的函数。

2）多层感知机：MLP

多层感知机的一个重要特点就是多层，我们将第一层称之为输入层，最后一层称之有输出层，中间的层称之为隐层。MLP 并没有规定隐层的数量，因此可以根据各自的需求选择合适的隐层层数。且对于输出层神经元的个数也没有限制。

MLP 神经网络结构模型如下,本文中只涉及了一个隐层，输入只有三个变量[x1,x2,x3][x1,x2,x3]和一个偏置量 bb，输出层有三个神经元。相比于感知机算法中的神经元模型对其进行了集成。



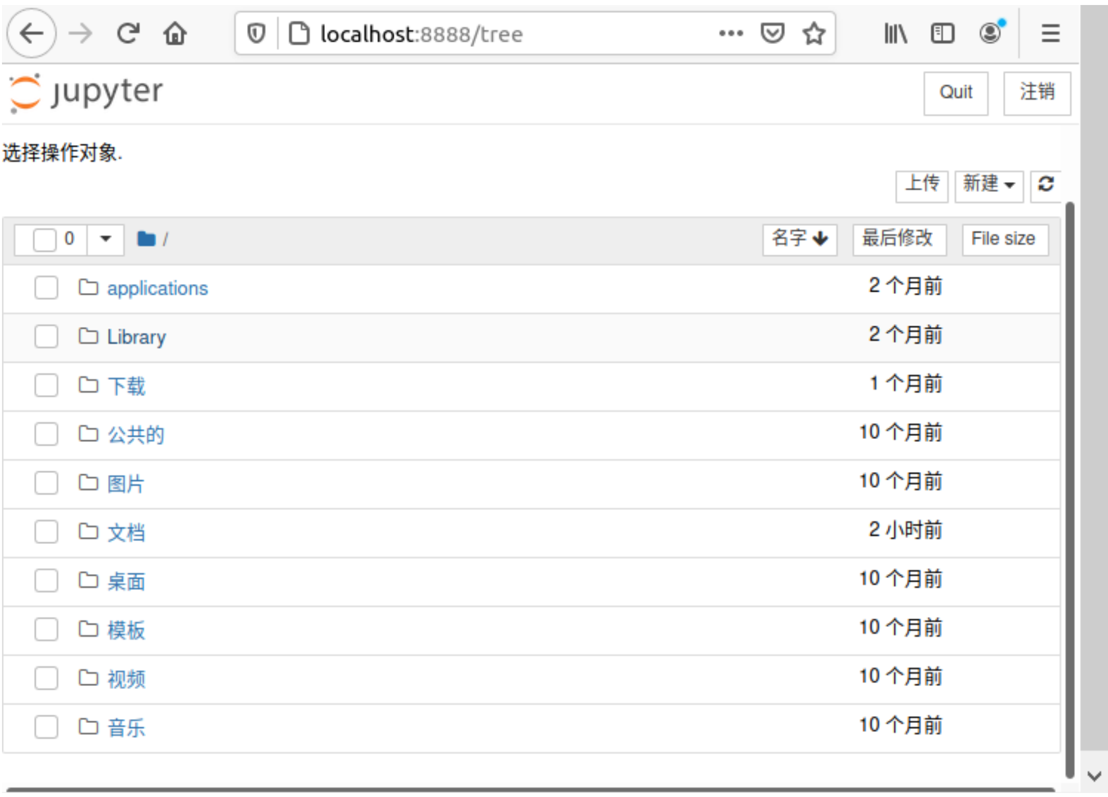


3. 实验环境

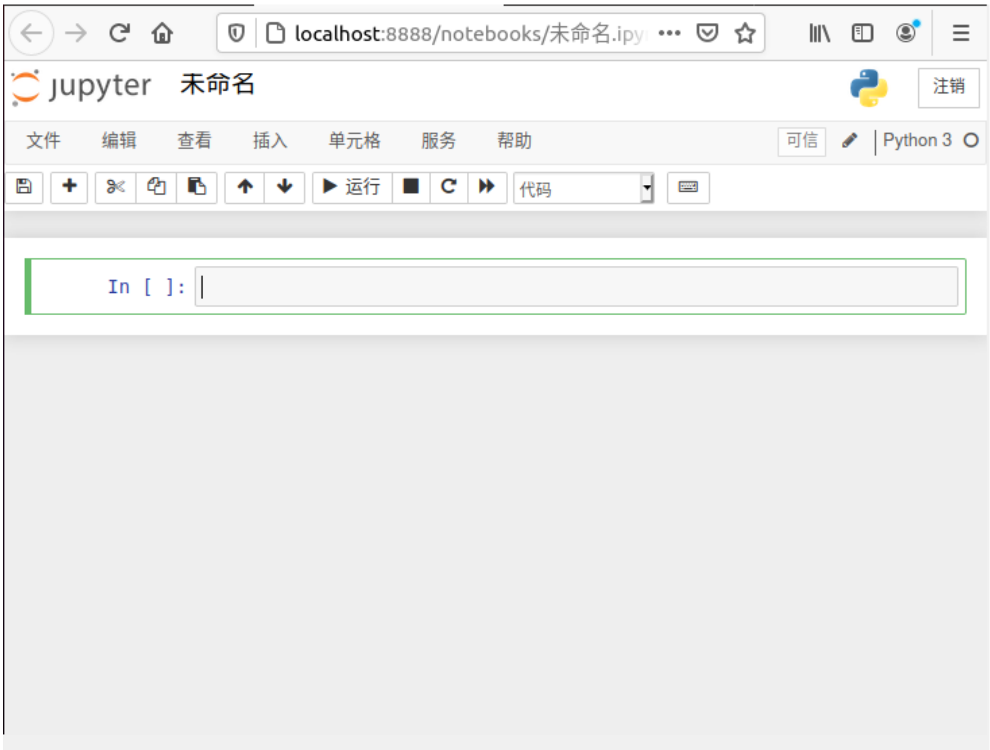
Ubuntu 20.04
Python 3.6
Jupyter notebook

4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面



2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件



5. 实操

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 特征归一化
4. 分割数据集
5. 绘制样例图片

```
#导入库
import numpy as np
from sklearn.datasets import load_digits
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import OneHotEncoder

#导入数据集
digits_dataset = load_digits()


# 特征归一化
X = minmax_scale(digits_dataset.data) # 特征逐维归一化
ohe = OneHotEncoder()
y = np.array(ohe.fit_transform(digits_dataset.target.reshape(-
1,1)).todense()) # 将类标签转变为 one hot 型变量
```

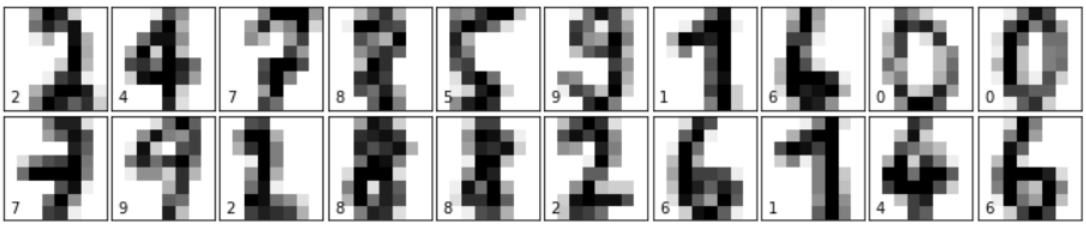
```python
#分割数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

np.random.seed(1)
shuffle_index = np.random.permutation(X_train.shape[0]) # 对训练集打乱顺序
X_train = X_train[shuffle_index, :]
y_train = y_train[shuffle_index, :]
# 绘制样例图片
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(10, 2))
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

for i in range(20):
    ax = fig.add_subplot(2, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(X_train[i,:].reshape(8,8), cmap=plt.cm.binary, interpolation='nearest')
    ax.text(0, 7, '{}'.format(np.argwhere(y_train[i,:]==1)[0][0]))

plt.show()
```



Step 2:MLP 模型

```python
class myMLP3():
    """
        This is a simple implementation for 3-layer neural network (i.e., input layer, hidden layer and output layer)
    """
    def __init__(self, input_layer_size=64, hidden_layer_size=100, output_layer_size=10,
                learning_rate=1, epochs=2000):
        """
```

```
            MLP3 initilization
        """
        self.input_layer_size = input_layer_size
        self.hidden_layer_size = hidden_layer_size
        self.output_layer_size = output_layer_size
        self.learning_rate = learning_rate
        self.epochs = epochs

        # 模型参数
        self.params = { 'W1': np.random.randn(self.hidden_layer_size, self.input_layer_size),
            'b1': np.zeros((self.hidden_layer_size, 1)),
            'W2': np.random.randn(self.output_layer_size, self.hidden_layer_size),
            'b2': np.zeros((self.output_layer_size, 1))}

        # 缓存变量：保留前向计算函数产生的汇聚值（非线性变换单元输入）和激励值（非线性变换单元输出）
        self.cache = {}

        # 梯度变量：保留反向传播函数计算得到的模型参数 W1,b1,W2,b2 的更新梯度
        self.grads = {}

    def sigmoid(self, z):
        """
            sigmoid function for nonlinear activation
        """
        a = 1 / (1 + np.exp(-z))
        return a

    def softmax(self, z):
        """
            softmax function for output
        """
        a = np.exp(z) / np.sum(np.exp(z), axis=0)
        return a

    def compute_multiclass_loss(self, Y, Y_hat):
        """
            loss function for multiclass classification
        """
        # 交叉熵损失
        L_sum = np.sum(np.multiply(Y, np.log(Y_hat)))
        sample_num = Y.shape[1]
```

```
        L = -(1/sample_num) * L_sum
        return L

    def feed_forward(self, X):
        """
            forward computation
        """
        # 前向计算：计算非线性截点的汇聚值（非线性变换单元输入）和激励值（非线性变换单元输出）
        self.cache['Z1'] = np.matmul(self.params["W1"], X) + self.params["b1"]
        self.cache['A1'] = self.sigmoid(self.cache['Z1'])

        self.cache['Z2'] = np.matmul(self.params["W2"], self.cache["A1"]) + self.params["b2"]
        self.cache['A2'] = self.softmax(self.cache['Z2'])

    def back_propagate(self, X, Y):
        """
            backward propagation
        """
        # 反向传播：计算模型参数 W1,b1,W2,b2 的更新梯度
        sample_num = X.shape[1]

        dZ2 = self.cache["A2"] - Y
        dW2 = (1./sample_num) * np.matmul(dZ2, self.cache["A1"].T)
        db2 = (1./sample_num) * np.sum(dZ2, axis=1, keepdims=True)

        dA1 = np.matmul(self.params["W2"].T, dZ2)
        dZ1 = dA1 * self.sigmoid(self.cache["Z1"]) * (1 - self.sigmoid(self.cache["Z1"]))
        dW1 = (1./sample_num) * np.matmul(dZ1, X.T)
        db1 = (1./sample_num) * np.sum(dZ1, axis=1, keepdims=True)

        self.grads = {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2}
        return self.grads

    def train(self, X, Y):
        """
            MLP3 training
        """
        for i in range(epochs):
            self.feed_forward(X)
            self.back_propagate(X, Y)
```

```
            # 参数更新：梯度下降法
            self.params['W2'] = self.params['W2'] - self.learning_rate
* self.grads['dW2']
            self.params['b2'] = self.params['b2'] - self.learning_rate
* self.grads['db2']
            self.params['W1'] = self.params['W1'] - self.learning_rate
* self.grads['dW1']
            self.params['b1'] = self.params['b1'] - self.learning_rate
* self.grads['db1']

            if (i % 20 == 0):
                print("Epoch", i, "cost: ", self.compute_multiclass_los
s(Y, self.cache["A2"]))
            elif (i==epochs-1):
                print("Epoch", epochs, "cost: ", self.compute_multiclas
s_loss(Y, self.cache["A2"]))
        #return self.params

    def predict(self, X):
        """
            MLP3 prediction
        """
        self.feed_forward(X)
        return np.argmax(self.cache["A2"], axis=0)

# 实验参数：选择不同的隐含层结点个数、学习率和训练循环迭代次数
hidden_layer_size = 100 # 隐含层结点个数
learning_rate = 1 # 学习率
epochs = 200 # BP 算法循环迭代次数

input_layer_size = X_train.shape[1]  # 输入层结点数，等于 64 （8*8 图像像
素）
output_layer_size = y_train.shape[1] # 输出层结点数，等于 10 （0,1,...,9 个
类别）

# 模型初始化及训练
mlp3 = myMLP3(input_layer_size, hidden_layer_size, output_layer_size, l
earning_rate, epochs) # 模型初始化
mlp3.train(X_train.T, y_train.T) # 模型训练
```

Step 3:模型测试

```
# 模型测试
predictions = mlp3.predict(X_test.T) # 预测标签
```

```
labels = np.argmax(y_test.T, axis=0) # 标签标签

print('confusion matrix: \n', confusion_matrix(labels, predictions)) # 输出分类混淆矩阵
print('\nclassification report: \n', classification_report(labels, predictions)) # 输出分类报告
```
Step 4：对比结果

```
# 实验参数：自选其他的图像，然后用学习得到的 MLP 分类器进行预测，比对预测结果与实际结果的一致性
new_idx = 0
assert new_idx in range(y.shape[0])
X_new = X[new_idx]
y_new = y[new_idx]
print('selected image category is %d' % np.where(y_new==1)[0][0])
plt.imshow(X_new.reshape(8,8), cmap=plt.cm.binary, interpolation='nearest')
plt.show()
```

```
selected image category is 0
```