实验六：BP神经网络算法

代码如下：

```python
import numpy as np
import random

class BPNeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.1):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate

        # 初始化权重和阈值（输入层到隐含层，隐含层到输出层）
        self.w1 = np.random.randn(input_size, hidden_size) * 0.01
        self.b1 = np.zeros((1, hidden_size))
        self.w2 = np.random.randn(hidden_size, output_size) * 0.01
        self.b2 = np.zeros((1, output_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_deriv(self, x):
        s = self.sigmoid(x)
        return s * (1 - s)

    def forward(self, x):
        self.hidden = self.sigmoid(np.dot(x, self.w1) + self.b1)
        self.output = self.sigmoid(np.dot(self.hidden, self.w2) + self.b2)
        return self.output

    def backward(self, x, y, output):
        # 计算输出层误差
        output_error = y - output
        delta_output = output_error * self.sigmoid_deriv(output)

        # 计算隐含层误差
        hidden_error = delta_output.dot(self.w2.T)
        delta_hidden = hidden_error * self.sigmoid_deriv(self.hidden)

        # 更新权重和阈值
        self.w2 += self.learning_rate * self.hidden.T.dot(delta_output)
        self.b2 += self.learning_rate * np.sum(delta_output, axis=0)
        self.w1 += self.learning_rate * x.T.dot(delta_hidden)
        self.b1 += self.learning_rate * np.sum(delta_hidden, axis=0)

    def train(self, X, y, epochs=1000, verbose=False):
        for epoch in range(epochs):
            output = self.forward(X)
            loss = np.mean(np.square(y - output))
            self.backward(X, y, output)
            if verbose and (epoch % 100 == 0):
                print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

```python
    def predict(self, X):
        output = self.forward(X)
        return np.argmax(output, axis=1) + 1  # 转换为1,2,3类别

def load_iris_data(filename):
    with open(filename, 'r') as f:
        data = [line.strip().split(',') for line in f]
    data = np.array(data, dtype=object)
    # 分离特征和标签
    features = data[:, :-1].astype(float)
    labels = data[:, -1]
    # 归一化特征到[0,1]
    features = (features - features.min(axis=0)) / (features.max(axis=0) -
features.min(axis=0))
    # 将标签映射为1,2,3
    label_map = {'Iris-setosa': 1, 'Iris-versicolor': 2, 'Iris-virginica': 3}
    labels = np.array([label_map[label] for label in labels])
    return features, labels

def split_dataset(features, labels):
    # 按类别分组（每组50个样本）
    setosa = features[labels==1], labels[labels==1]
    versicolor = features[labels==2], labels[labels==2]
    virginica = features[labels==3], labels[labels==3]

    # 训练集：每类前25个，测试集：每类后25个
    X_train = np.vstack((setosa[0][:25], versicolor[0][:25], virginica[0][:25]))
    y_train = np.hstack((setosa[1][:25], versicolor[1][:25], virginica[1][:25]))
    X_test = np.vstack((setosa[0][25:], versicolor[0][25:], virginica[0][25:]))
    y_test = np.hstack((setosa[1][25:], versicolor[1][25:], virginica[1][25:]))

    # 转换为独热编码
    def one_hot_encode(labels):
        n = len(labels)
        one_hot = np.zeros((n, 3))
        one_hot[np.arange(n), labels-1] = 1
        return one_hot
    y_train_onehot = one_hot_encode(y_train)
    y_test_onehot = one_hot_encode(y_test)
    return X_train, y_train_onehot, X_test, y_test

def run_experiment(learning_rate=0.1, epochs=1000):
    features, labels = load_iris_data('Iris.txt.txt')
    X_train, y_train, X_test, y_test = split_dataset(features, labels)

    nn = BPNeuralNetwork(
        input_size=4,
        hidden_size=10,
        output_size=3,
        learning_rate=learning_rate
    )
    nn.train(X_train, y_train, epochs=epochs, verbose=False)
    y_pred = nn.predict(X_test)
    accuracy = np.mean(y_pred == y_test) * 100
    return accuracy
```

```python
if __name__ == "__main__":
    np.random.seed(42)
    learning_rate = 0.1  # 学习率
    epochs = 1000
    accuracies = []

    for i in range(10):
        acc = run_experiment(learning_rate, epochs)
        accuracies.append(acc)
        print(f"Run {i+1}, Accuracy: {acc:.2f}%")

    avg_accuracy = np.mean(accuracies)
    std_accuracy = np.std(accuracies)
    print(f"\nAverage Accuracy: {avg_accuracy:.2f}%")
    print(f"Standard Deviation: {std_accuracy:.2f}%")
```

数据处理：

- 读取鸢尾花数据集，将特征归一化到 [0,1] 区间，标签映射为 1、2、3。
- 按类别划分训练集和测试集：每类前 25 个样本为训练集，后 25 个为测试集，确保训练集和测试集各 75 样本。
- 将标签转换为独热编码（One-Hot Encoding）以适配神经网络输出。

BP 神经网络结构：

- **输入层**：4 个神经元（对应 4 个特征）。
- **隐含层**：10 个神经元，激活函数为 Sigmoid。
- **输出层**：3 个神经元（对应 3 个类别），激活函数为 Sigmoid，输出值表示属于各品种的概率。

关键函数：

- `forward()`：前向传播计算隐含层和输出层激活值。
- `backward()`：反向传播计算误差梯度，更新权重和阈值（学习率设为 0.1）。
- `train()`：迭代训练网络，使用均方误差（MSE）作为损失函数。
- `predict()`：将输出概率转换为类别标签（取最大值对应的类别）。