蠓虫分类问题的神经网络方法

区分两类蠓虫的方案

设计一个简单的**BP**神经网络，通过对输入的样本数据进行学习后，该神经网络可将待测试样本映射为**0-1**间的一个数字，越靠近**0**则样本为**Apf**的概率越大，越靠近**1**则样本为**Af**的概率越大。以下是该**BP**网络的搭建过程：

1. 首先传入原始数据，并为每个原始数据记录标签（Af记为1、Apf记为0）

```
% 原始数据
input_data = [1.24 1.72; 1.36 1.74; 1.38 1.64; 1.38 1.82; 1.38 1.90; 1.40 1.70; 1.48
1.82; 1.54 1.82; 1.56 2.08; 1.14 1.78; 1.18 1.96; 1.20 1.86; 1.26 2.00; 1.28 2.00; 1.30
1.96]; % 前9组为Af，后9组为Apf
target_data = [ones(1, 9), zeros(1, 6)]; % Af对应1，Apf对应0
```

2. 设定BP神经网络参数，随机初始化权重与偏置（随着迭代各参数会不断优化）

```
% 设置神经网络参数
epochs = 10000; % 迭代次数
hidden_size = 5; % 隐藏层神经元数量

% 初始化权重和偏置
input_size = size(input_data, 2); % 输入数据的维度
output_size = 1; % 输出数据的维度
hidden_weights = rand(hidden_size, input_size); % 隐藏层权重，随机初始化
output_weights = rand(output_size, hidden_size); % 输出层权重，随机初始化
hidden_bias = rand(hidden_size, 1); % 隐藏层偏置，随机初始化
output_bias = rand(output_size, 1); % 输出层偏置，随机初始化

% 存储训练误差
errors = zeros(1, epochs); % 用于存储每次迭代的训练误差
```

3. 训练BP神经网路

- 学习率是一个非常关键的超参数。为了使训练更加稳定，可以在训练中逐渐降低学习率。方便起见，当训练轮次到达1000轮后，将学习率由0.1下降至0.05。
- bp神经网络使用的激活函数是非线性变换函数——Sigmoid函数（又称S函数）激活函数的作用是增加非线性因素，解决线性模型表达能力不足的缺陷。

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad sigmoid'(x) = sigmoid(x) * (1 - sigmoid(x))$$

- 根据以上两个函数可确定前向传播、反向传播算法步骤，并记录误差。

```
for epoch = 1:epochs
    % 根据迭代次数调整学习率
    if epochs < 1000
        learning_rate = 0.1;
```

```
    else
        learning_rate = 0.05;
    end
    err = 0; % 初始化累计误差
    for i = 1:size(input_data, 1) % 遍历所有训练样本
        % 前向传播
        hidden_output = 1 ./ (1 + exp(-(hidden_weights * input_data(i, :)' +
hidden_bias))); % 计算隐藏层输出
        output = 1 ./ (1 + exp(-(output_weights * hidden_output + output_bias))); % 计算
神经网络输出

        % 反向传播
        output_delta = (output - target_data(i)) * output * (1 - output); % 计算输出层误
差
        hidden_delta = (output_weights' * output_delta) .* hidden_output .* (1 -
hidden_output); % 计算隐藏层误差

        % 更新权重和偏置
        output_weights = output_weights - learning_rate * output_delta * hidden_output';
% 更新输出层权重
        hidden_weights = hidden_weights - learning_rate * hidden_delta * input_data(i,
:); % 更新隐藏层权重
        output_bias = output_bias - learning_rate * output_delta; % 更新输出层偏置
        hidden_bias = hidden_bias - learning_rate * hidden_delta; % 更新隐藏层偏置

        % 累计误差
        err = err + (output - target_data(i))^2; % 计算累计误差
    end
    errors(epoch) = err / size(input_data, 1); % 计算平均训练误差
end
```

4. 绘制误差变化曲线

```
figure;
plot(1:epochs, errors); % 绘制误差变化曲线
xlabel('Epochs');
ylabel('Error');
title('Training Error Curve');
```

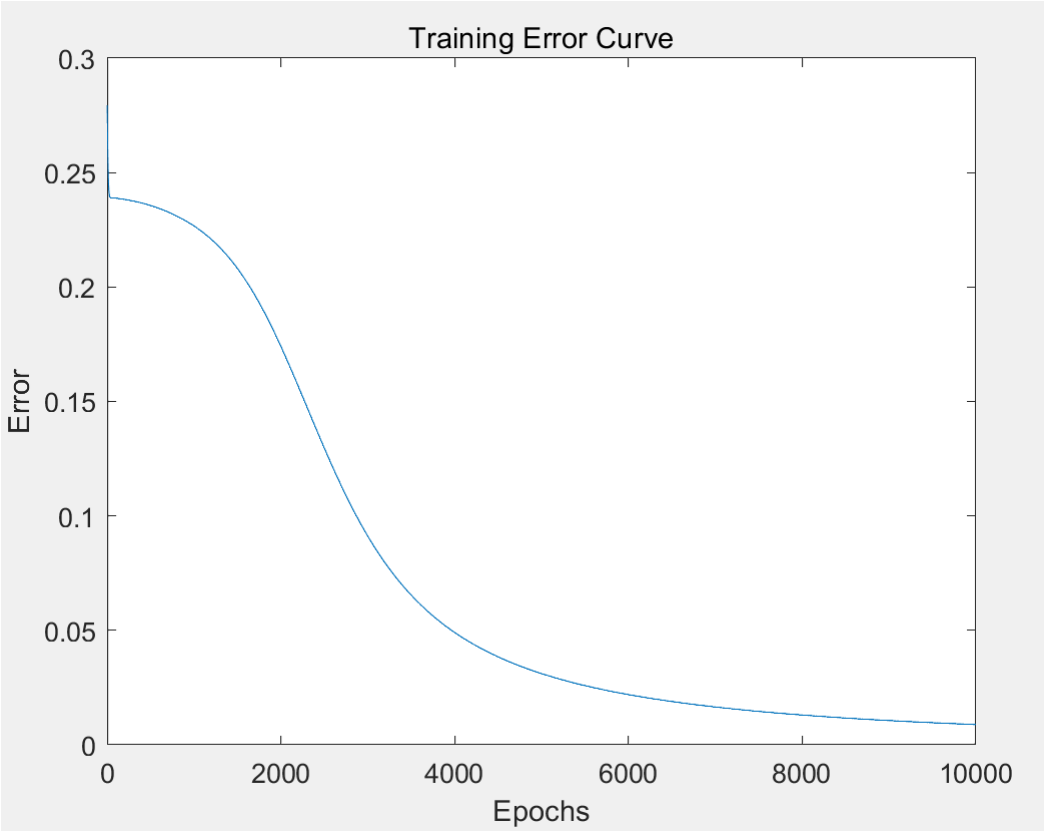
5. 预测结果

```
% 对给定的三个样本进行预测并输出结果
test_samples = [1.24 1.80; 1.28 1.84; 1.40 2.04]; % 测试样本
predictions = strings(1, size(test_samples, 1)); % 用于存储预测结果
for i = 1:size(test_samples, 1) % 遍历所有测试样本
    hidden_output = 1 ./ (1 + exp(-(hidden_weights * test_samples(i, :)' +
hidden_bias))); % 计算隐藏层输出
    output = 1 ./ (1 + exp(-(output_weights * hidden_output + output_bias))); % 计算神经
网络输出
    predictions(i) = output;
end

disp("预测结果为:");
disp(predictions);
```

结果分析：

1. 训练误差曲线：



神经网络刚开始学习时，误差迅速下降，随后逐渐减缓衰减速度，最终逐渐收敛为0。

2. 预测结果

	样本1	样本2	样本3
测试1	0.50374	0.59563	0.31481
测试2	0.49866	0.59525	0.32849
测试3	0.50093	0.59746	0.32736
测试4	0.50095	0.59449	0.32058
测试5	0.49948	0.59549	0.32625

根据结果，第一个样本的品种不好判断；推测第二个样本为Af，第三个样本为Apf。