# 《机器学习与深度学习》课程

# 实 验 报 告



**姓 名： 金家耀**

**专 业：**  人工智能

**学 号： 1193210320**

**江南大学人工智能与计算机学院**

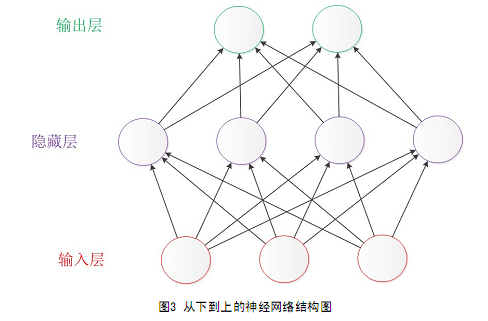
# BP神经网络

**1实验目的**

BP神经网络是一种按误差[反向传播](https://baike.baidu.com/item/%E5%8F%8D%E5%90%91%E4%BC%A0%E6%92%AD/53364789?fromModule=lemma_inlink" \t "_blank)(简称误差反传)训练的多层前馈神经网络，具有输入层、隐藏层和输出层，具有任意复杂的模式分类能力和优良的多维函数映射能力，在许多实际工程中有着广泛的应用。本实验的目的在于加深学生对BP神经网络的理解，理解多层感知器用于分类的原理和方法，特别是解决非线性多类别分类问题，利用实际数据进行分类处理。

**2实验原理**

多层感知器网络的输入层对应样本的特征输入，输出层对应类别，中间隐含层的加入使得网络能够表达非线性的映射关系，利用误差反向传播（BP）算法进行网络训练。



**隐含层**

**输入层**

**输出层**

基本BP算法包括信号的前向传播和误差的反向传播两个过程。即[计算误差](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E8%AF%AF%E5%B7%AE/19161519?fromModule=lemma_inlink" \t "_blank)输出时按从输入到输出的方向进行，而调整[权值](https://baike.baidu.com/item/%E6%9D%83%E5%80%BC/170585?fromModule=lemma_inlink" \t "_blank)和阈值则从输出到输入的方向进行。正向传播时，[输入信号](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5%E4%BF%A1%E5%8F%B7/5455302?fromModule=lemma_inlink" \t "_blank)通过隐含层作用于输出节点，经过[非线性变换](https://baike.baidu.com/item/%E9%9D%9E%E7%BA%BF%E6%80%A7%E5%8F%98%E6%8D%A2/54688226?fromModule=lemma_inlink" \t "_blank)，产生[输出信号](https://baike.baidu.com/item/%E8%BE%93%E5%87%BA%E4%BF%A1%E5%8F%B7/5455317?fromModule=lemma_inlink" \t "_blank)，若实际输出与期望输出不相符，则转入误差的反向[传播过程](https://baike.baidu.com/item/%E4%BC%A0%E6%92%AD%E8%BF%87%E7%A8%8B/1195094?fromModule=lemma_inlink" \t "_blank)。误差反传是将[输出误差](https://baike.baidu.com/item/%E8%BE%93%E5%87%BA%E8%AF%AF%E5%B7%AE/56125485?fromModule=lemma_inlink" \t "_blank)通过隐含层向输入层逐层反传，并将误差分摊给各层所有单元，以从各层获得的[误差信号](https://baike.baidu.com/item/%E8%AF%AF%E5%B7%AE%E4%BF%A1%E5%8F%B7/56308361?fromModule=lemma_inlink" \t "_blank)作为调整各单元权值的依据。通过调整[输入节点](https://baike.baidu.com/item/%E8%BE%93%E5%85%A5%E8%8A%82%E7%82%B9/6893408?fromModule=lemma_inlink" \t "_blank)与隐层节点的联接强度和隐层节点与输出节点的联接强度以及阈值，使误差沿[梯度方向](https://baike.baidu.com/item/%E6%A2%AF%E5%BA%A6%E6%96%B9%E5%90%91/56288830?fromModule=lemma_inlink" \t "_blank)下降，经过反复学习训练，确定与最小误差相对应的[网络参数](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%8F%82%E6%95%B0/4474943?fromModule=lemma_inlink" \t "_blank)(权值和阈值)，训练即告停止。此时经过训练的神经网络即能对类似样本的输入信息，自行处理输出误差最小的经过非线形转换的信息。

BP算法的具体内容和相关公式推导可参见已在课程群中共享的课件。

**3实验内容和要求**

利用Iris数据集进行实验。该数据集以鸢尾花的特征作为数据来源，由3种不同类型的鸢尾花的50个样本数据构成。其中的第一个种类与另外两个种类是线性可分离的，后两个种类是非线性可分离的。

样本特征（属性）：

Sepal.Length（花萼长度），单位是cm;

Sepal.Width（花萼宽度），单位是cm;

Petal.Length（花瓣长度），单位是cm;

Petal.Width（花瓣宽度），单位是cm;

种类：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾）、Iris Virginica（维吉尼亚鸢尾）。

实验要求：

（1）利用Fisher线性判别或感知器算法验证“第一个种类与另外两个种类是线性可分离的，后两个种类是非线性可分离的”

（2）构建BP网络进行分类实验，并通过10次10折交叉验证给出识别率。

（3）调整BP网络的隐藏层数量，观察识别率的变化。

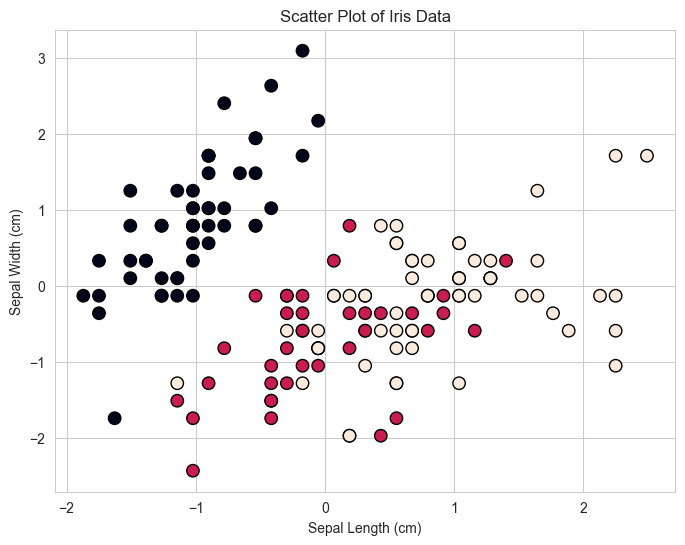
（4）调整BP网络的隐藏层神经元数量，观察识别率的变化。

**4实验代码和结果**

实验代码已上传至 <https://github.com/shinejjy/MachineLearningAndDeepLearning/sy5>

1. 读取Iris数据集

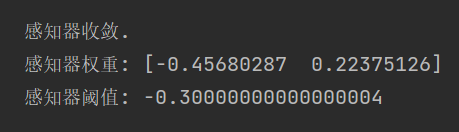
# 数据准备  
# 加载Iris数据集  
iris = load\_iris()  
X, y = iris.data, iris.target  
  
# 数据标准化  
scaler = StandardScaler()  
X = scaler.fit\_transform(X)  
X.shape, y.shape

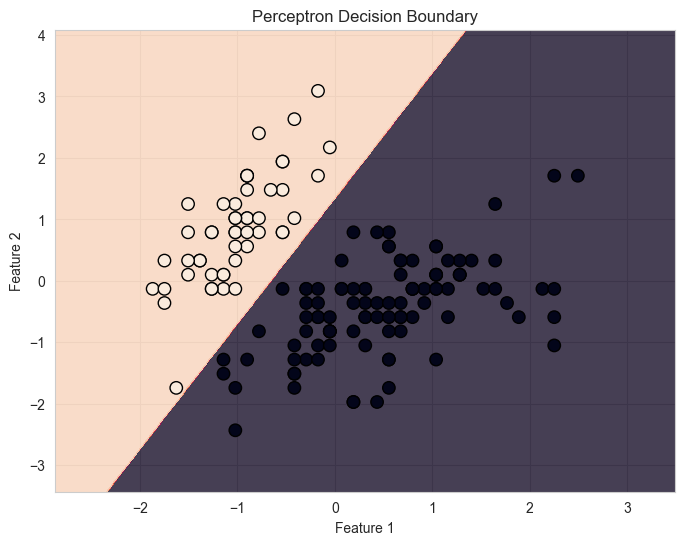


读取鸢尾花数据集后，将数据的前两个特征可视化在二维平面内。

1. 使用感知器算法验证第一类样本与其他两类样本线性可分。

# 把第一类设为1，其余两类设为-1  
y\_1 = np.where(y == 0, 1, -1)  
X\_1 = X[:, :2]  
  
# 初始化感知器参数  
w = np.zeros(X\_1.shape[1])  
b = 0  
learning\_rate = 0.1  
max\_epochs = 1000 # 设置最大迭代次数  
  
# 训练感知器  
converged = False  
for epoch in range(max\_epochs):  
 misclassified = False  
 for xi, target in zip(X\_1, y\_1):  
 if target \* (np.dot(xi, w) + b) <= 0:  
 w += learning\_rate \* target \* xi  
 b += learning\_rate \* target  
 misclassified = True  
 if not misclassified:  
 converged = True  
 break  
  
# 打印感知器参数  
if converged:  
 print("感知器收敛.")  
 print("感知器权重:", w)  
 print("感知器阈值:", b)  
else:  
 print("感知器未收敛，达到最大迭代次数.")

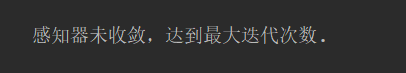


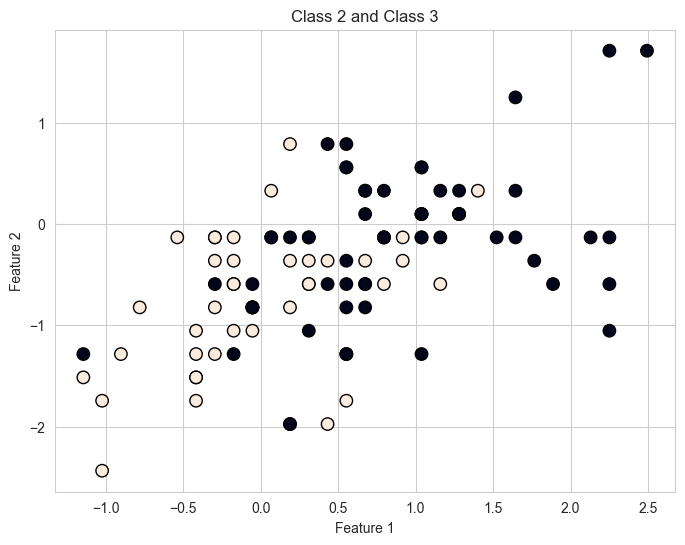


为了方便样本的可视化观察，同样地，在本次实验中只选取了前两个特征作为实验数据。为了适应感知器算法，我将第一类鸢尾花的标签设置为1，将其余类别的鸢尾花的标签设置为-1。从结果中可以得出，第一类鸢尾花可以和其余类别的鸢尾花线性可分，并做出二维平面的可视化。

1. 使用感知器算法验证后两类样本非线性可分

# 把第二类设为1，其余设为-1  
y\_2 = y[y != 0]  
X\_2 = X[y != 0, :2]  
y\_2 = np.where(y\_2 != 1, -1, 1)  
  
# 初始化感知器参数  
w = np.zeros(X\_2.shape[1])  
b = 0  
learning\_rate = 0.1  
max\_epochs = 1000 # 设置最大迭代次数  
  
# 训练感知器  
converged = False  
for epoch in range(max\_epochs):  
 misclassified = False  
 for xi, target in zip(X\_2, y\_2):  
 if target \* (np.dot(xi, w) + b) <= 0:  
 w += learning\_rate \* target \* xi  
 b += learning\_rate \* target  
 misclassified = True  
 if not misclassified:  
 converged = True  
 break  
  
# 打印感知器参数  
if converged:  
 print("感知器收敛.")  
 print("感知器权重:", w)  
 print("感知器阈值:", b)  
else:  
 print("感知器未收敛，达到最大迭代次数.")





同理，选取后两类鸢尾花的前两维特征作为实验数据，并将第二种鸢尾花的标签设置成1，将第三种鸢尾花的标签设置成1。从结果中可以看出，感知器无法在有限的时间内收敛，这就说明后两个类别的鸢尾花是非线性可分的。

1. sigmoid函数与导函数

def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
def sigmoid\_derivative(x):  
 return x \* (1 - x)

1. softmax函数

def softmax(x):  
 exp\_x = np.exp(x - np.max(x, axis=1, keepdims=True))  
 return exp\_x / np.sum(exp\_x, axis=1, keepdims=True)

1. 交叉熵损失函数

def cross\_entropy\_loss(y\_true, y\_pred):  
 m = y\_true.shape[0]  
 log\_likelihood = -np.log(y\_pred[range(m), y\_true])  
 loss = np.sum(log\_likelihood) / m  
 return loss

1. 独热向量编码

def one\_hot\_encode(y, num\_classes):  
 m = y.shape[0]  
 one\_hot = np.zeros((m, num\_classes))  
 one\_hot[np.arange(m), y] = 1  
 return one\_hot

1. 构建与训练BP神经网络

def train\_neural\_network(X, y, hidden\_layer\_sizes, learning\_rate, epochs, seed=42):  
 # 设置种子  
 np.random.seed(seed)  
  
 input\_size = X.shape[1]  
 output\_size = len(np.unique(y))  
 num\_layers = len(hidden\_layer\_sizes) + 1  
 losses = []  
  
 # 初始化权重和偏差  
 weights = [np.random.randn(input\_size, hidden\_layer\_sizes[0])]  
 biases = [np.zeros((1, hidden\_layer\_sizes[0]))]  
 for i in range(1, num\_layers - 1):  
 weights.append(np.random.randn(hidden\_layer\_sizes[i - 1], hidden\_layer\_sizes[i]))  
 biases.append(np.zeros((1, hidden\_layer\_sizes[i])))  
 weights.append(np.random.randn(hidden\_layer\_sizes[-1], output\_size))  
 biases.append(np.zeros((1, output\_size)))  
  
 for epoch in range(epochs):  
 # 前向传播  
 layer\_outputs = [X]  
 for i in range(num\_layers):  
 layer\_inputs = layer\_outputs[-1] @ weights[i] + biases[i]  
 if i == num\_layers - 1:  
 layer\_outputs.append(softmax(layer\_inputs))  
 else:  
 layer\_outputs.append(sigmoid(layer\_inputs))  
  
 # 计算损失  
 loss = cross\_entropy\_loss(y, layer\_outputs[-1])  
 losses.append(loss)  
  
 # 反向传播  
 errors = [layer\_outputs[-1] - one\_hot\_encode(y, output\_size)]  
 for i in range(num\_layers - 2, -1, -1):  
 errors.insert(0, errors[0] @ weights[i + 1].T \* sigmoid\_derivative(layer\_outputs[i + 1]))  
  
 # 更新权重和偏差  
 for i in range(num\_layers):  
 weights[i] -= learning\_rate \* layer\_outputs[i].T @ errors[i]  
 biases[i] -= learning\_rate \* np.sum(errors[i], axis=0, keepdims=True)  
  
 if epoch % 100 == 0:  
 print(f"Epoch {epoch}, Loss: {loss:.4f}")  
  
 return weights, biases, losses

实现了一个简单的神经网络训练过程。首先，通过给定的输入特征数、输出类别数以及隐藏层结构，初始化了神经网络的权重和偏差。接下来，通过指定的训练迭代次数，循环进行训练过程。

在每一轮训练中，首先进行前向传播，通过矩阵运算和激活函数计算每个神经元的输出。然后，利用交叉熵损失函数计算预测结果与实际标签之间的损失。接着进行反向传播，通过梯度下降法计算每个参数对损失的梯度。最后，根据梯度和学习率更新权重和偏差。

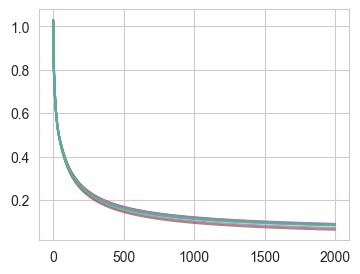
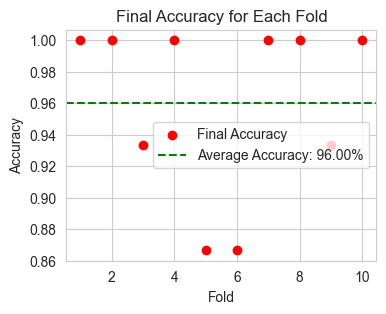
训练过程中，每隔100轮迭代会打印当前迭代轮次和损失值，以便监控训练过程。最终，函数返回训练得到的权重矩阵、偏差矩阵以及损失值列表。

1. BP神经网路预测

def predict(X, weights, biases):  
 num\_layers = len(weights)  
 layer\_outputs = [X]  
 for i in range(num\_layers):  
 layer\_inputs = layer\_outputs[-1] @ weights[i] + biases[i]  
 if i == num\_layers - 1:  
 layer\_outputs.append(softmax(layer\_inputs))  
 else:  
 layer\_outputs.append(sigmoid(layer\_inputs))  
 return np.argmax(layer\_outputs[-1], axis=1)

1. 隐藏层神经元个数：1

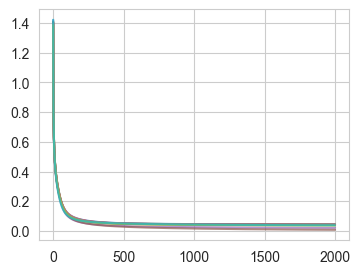
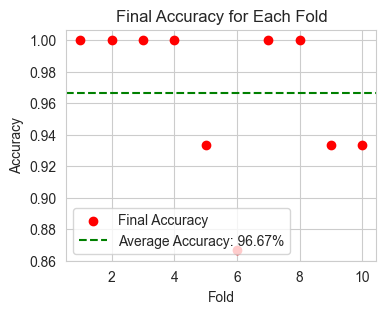
from matplotlib import pyplot as plt  
  
# 设置超参数  
learning\_rate = 0.01  
epochs = 2000  
hidden\_layer\_sizes = [1] # 调整隐藏层的数量和神经元数量  
  
# 10次10折交叉验证  
kf = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=42)  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
  
accuracies = []  
  
for idx, (train\_index, test\_index) in enumerate(kf.split(X, y)):  
 print(f"Fold {idx + 1}:")  
 X\_train, X\_test = X[train\_index], X[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # 训练神经网络  
 weights, biases, losses = train\_neural\_network(X\_train, y\_train, hidden\_layer\_sizes, learning\_rate, epochs)  
  
 # 可视化学习曲线  
 plt.plot(range(epochs), losses, label=f'Fold {idx + 1}', alpha=0.7) # 设置alpha以调整线的透明度  
  
 # 预测  
 predictions = predict(X\_test, weights, biases)  
  
 # 计算准确率  
 accuracy = np.mean(predictions == y\_test)  
 print(f"Fold {idx + 1} Accuracy: {accuracy: .2%}\n")  
 accuracies.append(accuracy)  
  
# 打印平均准确率  
average\_accuracy = np.mean(accuracies)  
print(f"Average Accuracy: {average\_accuracy:.2%}")  
  
# 可视化每个fold的最终准确率  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
plt.scatter(range(1, 11), accuracies, color='red', marker='o', label='Final Accuracy')  
plt.axhline(y=average\_accuracy, color='green', linestyle='--', label=f'Average Accuracy: {average\_accuracy:.2%}')  
plt.xlabel('Fold')  
plt.ylabel('Accuracy')  
plt.title('Final Accuracy for Each Fold')  
plt.legend()  
plt.show()

将神经网路的隐藏层数设置为1层，将神经元的个数设置成1，经过学习率为0.01的2000轮迭代后，最终10折交叉验证的平均准确率达到96.00%。

1. 隐藏层神经元个数：6

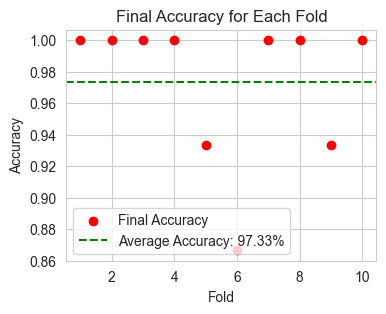
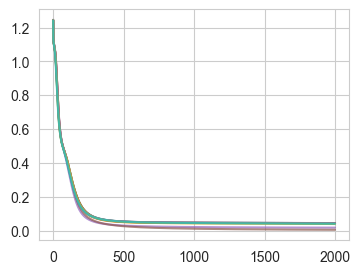
from matplotlib import pyplot as plt  
  
# 设置超参数  
learning\_rate = 0.01  
epochs = 2000  
hidden\_layer\_sizes = [6] # 调整隐藏层的数量和神经元数量  
  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
  
accuracies = []  
  
for idx, (train\_index, test\_index) in enumerate(kf.split(X, y)):  
 print(f"Fold {idx + 1}:")  
 X\_train, X\_test = X[train\_index], X[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # 训练神经网络  
 weights, biases, losses = train\_neural\_network(X\_train, y\_train, hidden\_layer\_sizes, learning\_rate, epochs)  
  
 # 可视化学习曲线  
 plt.plot(range(epochs), losses, label=f'Fold {idx + 1}', alpha=0.7) # 设置alpha以调整线的透明度  
  
 # 预测  
 predictions = predict(X\_test, weights, biases)  
  
 # 计算准确率  
 accuracy = np.mean(predictions == y\_test)  
 print(f"Fold {idx + 1} Accuracy: {accuracy: .2%}\n")  
 accuracies.append(accuracy)  
  
# 打印平均准确率  
average\_accuracy = np.mean(accuracies)  
print(f"Average Accuracy: {average\_accuracy:.2%}")  
  
# 可视化每个fold的最终准确率  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
plt.scatter(range(1, 11), accuracies, color='red', marker='o', label='Final Accuracy')  
plt.axhline(y=average\_accuracy, color='green', linestyle='--', label=f'Average Accuracy: {average\_accuracy:.2%}')  
plt.xlabel('Fold')  
plt.ylabel('Accuracy')  
plt.title('Final Accuracy for Each Fold')  
plt.legend()  
plt.show()

调整神经元的个数，将神经网路的隐藏层数设置为1层，将神经元的个数设置成6，经过学习率为0.01的2000轮迭代后，最终10折交叉验证的平均准确率达到96.67%。

1. 隐藏层神经元个数：2，3

from matplotlib import pyplot as plt  
  
# 设置超参数  
learning\_rate = 0.01  
epochs = 2000  
hidden\_layer\_sizes = [2, 3] # 调整隐藏层的数量和神经元数量  
  
# 10次10折交叉验证  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
  
accuracies = []  
  
for idx, (train\_index, test\_index) in enumerate(kf.split(X, y)):  
 print(f"Fold {idx + 1}:")  
 X\_train, X\_test = X[train\_index], X[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # 训练神经网络  
 weights, biases, losses = train\_neural\_network(X\_train, y\_train, hidden\_layer\_sizes, learning\_rate, epochs)  
  
 # 可视化学习曲线  
 plt.plot(range(epochs), losses, label=f'Fold {idx + 1}', alpha=0.7) # 设置alpha以调整线的透明度  
  
 # 预测  
 predictions = predict(X\_test, weights, biases)  
  
 # 计算准确率  
 accuracy = np.mean(predictions == y\_test)  
 print(f"Fold {idx + 1} Accuracy: {accuracy: .2%}\n")  
 accuracies.append(accuracy)  
  
# 打印平均准确率  
average\_accuracy = np.mean(accuracies)  
print(f"Average Accuracy: {average\_accuracy:.2%}")  
  
# 可视化每个fold的最终准确率  
plt.figure(figsize=(4, 3)) # 新建一个图，设置图的大小  
plt.scatter(range(1, 11), accuracies, color='red', marker='o', label='Final Accuracy')  
plt.axhline(y=average\_accuracy, color='green', linestyle='--', label=f'Average Accuracy: {average\_accuracy:.2%}')  
plt.xlabel('Fold')  
plt.ylabel('Accuracy')  
plt.title('Final Accuracy for Each Fold')  
plt.legend()  
plt.show()



调整隐藏层的个数，将神经网路的隐藏层数设置为2层，将神经元的个数分别设置成2和3，经过学习率为0.01的2000轮迭代后，最终10折交叉验证的平均准确率达到97.33%。

1. 总结

在进行神经网络的调参过程中，通过对隐藏层数、神经元个数以及学习率等关键参数进行调整，取得了不同模型性能的提升。首先，在隐藏层数为1、神经元个数为1的情况下，通过学习率为0.01的2000轮迭代，取得了96.00%的准确率。随后，增加神经元个数至6，准确率进一步提高至96.67%。最后，在隐藏层数为2、神经元个数分别为2和3的情况下，通过相同的学习率和迭代次数，实现了97.33%的平均准确率。这说明对于该问题，增加神经元个数和隐藏层数均有助于提升神经网络模型的性能。

**5实验心得**

在进行神经网络调参实验的过程中，我深刻体会到了调整不同超参数对模型性能的影响。首先，单层神经网络在适当的学习率下表现出色，通过适当的参数选择，取得了令人满意的准确率。其次，在增加神经元个数的情况下，模型的表现有所提升，但也需要注意过大的神经元个数可能导致过拟合。最后，通过增加隐藏层数，模型在一定程度上实现了更好的性能，但需要权衡计算成本和性能提升。

总体而言，深入理解神经网络结构和不同参数的作用，通过反复调整和实验，可以更好地优化模型性能。这个过程既需要对深度学习原理的理解，也需要通过实验不断调整参数，从而更好地理解模型行为和调参策略。这次实验为我提供了更多的实践经验，使我对神经网络的调参和性能优化有了更深入的认识。