# 《人工智能》实验四报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 俞家宝 | 学号 | 2021117338 |
| 实验地点 | 实验室321 | 实验日期 | 2023/11/6 |

**一、实验内容**

# Build a backpropngntion network in a language of your choice and run it on MNIST

Dataset。MNIST Dataset can be download from [***http://yann.lecun.com/exdb/mnist/.***](http://yann.lecun.com/exdb/mnist/.)

**（使用你喜欢的语言实现BP神经网络，并在MNIST手写数据集上进行训练和测试）**

1. **实验原理**

**Back Propagation是建立在梯度下降算法基础上，适用多层神经网络的参数训练方法。由于隐藏层节点的预测误差无法直接计算,因此,反向传播算法直接利用输出层节点的预测误差反向估计上一层隐藏节点的预测误差,即从后往前逐层从输出层把误差反向传播到输入层,从而实现对链接权重调整**

1. **实验过程以及结果分析**

**代码:**

import numpy as np  
# 定义激活函数  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))

# 定义BP神经网络类  
class NeuralNetwork:  
 def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):  
 self.input\_size = input\_size  
 self.hidden\_size = hidden\_size  
 self.output\_size = output\_size  
  
 # 随机初始化权重  
 self.W1 = np.random.randn(self.input\_size, self.hidden\_size)  
 self.b1 = np.zeros((1, self.hidden\_size))  
 self.W2 = np.random.randn(self.hidden\_size, self.output\_size)  
 self.b2 = np.zeros((1, self.output\_size))  
  
 # 前向传播  
 def forward(self, X):  
 self.z1 = np.dot(X, self.W1) + self.b1  
 self.a1 = sigmoid(self.z1)  
 self.z2 = np.dot(self.a1, self.W2) + self.b2  
 self.a2 = sigmoid(self.z2)  
 return self.a2  
  
 # 反向传播  
 def backward(self, X, y, learning\_rate):  
 m = X.shape[0]  
  
 # 计算输出层的误差  
 delta2 = self.a2 - y  
  
 # 计算隐藏层的误差  
 delta1 = np.dot(delta2, self.W2.T) \* self.a1 \* (1 - self.a1)  
  
 # 更新权重和偏置  
 dW2 = np.dot(self.a1.T, delta2) / m  
 db2 = np.sum(delta2, axis=0) / m  
 dW1 = np.dot(X.T, delta1) / m  
 db1 = np.sum(delta1, axis=0) / m  
  
 self.W2 -= learning\_rate \* dW2  
 self.b2 -= learning\_rate \* db2  
 self.W1 -= learning\_rate \* dW1  
 self.b1 -= learning\_rate \* db1  
  
 # 训练模型  
 def train(self, X, y, num\_epochs, learning\_rate):  
 for epoch in range(num\_epochs):  
 # 前向传播  
 output = self.forward(X)  
  
 # 反向传播  
 self.backward(X, y, learning\_rate)  
  
 # 计算损失  
 loss = np.mean(-y \* np.log(output) - (1 - y) \* np.log(1 - output))  
  
 # 每隔一段时间输出损失  
 if epoch % 100 == 0:  
 print(f"Epoch {epoch}, Loss: {loss}")  
  
 # 预测  
 def predict(self, X):  
 output = self.forward(X)  
 predictions = np.round(output)  
 return predictions  
  
  
# 读取数据  
def load\_data(image\_file, label\_file):  
 with open(label\_file, 'rb') as f:  
 labels = np.frombuffer(f.read(), dtype=np.uint8, offset=8)  
 with open(image\_file, 'rb') as f:  
 images = np.frombuffer(f.read(), dtype=np.uint8, offset=16).reshape(len(labels), -1)  
 return images, labels  
  
# 加载训练集和测试集数据  
train\_images, train\_labels = load\_data('train-images.idx3-ubyte', 'train-labels.idx1-ubyte')  
test\_images, test\_labels = load\_data('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte')  
  
# 数据预处理  
train\_images = train\_images / 255.0  
test\_images = test\_images / 255.0  
  
# 将标签转换为独热编码  
num\_classes = 10  
train\_labels = np.eye(num\_classes)[train\_labels]  
test\_labels = np.eye(num\_classes)[test\_labels]  
  
# 创建并训练神经网络模型  
input\_size = train\_images.shape[1]  
hidden\_size = 64  
output\_size = num\_classes  
numepochs = 1000  
learning\_rate = 0.1  
  
model = NeuralNetwork(input\_size, hidden\_size, output\_size)  
model.train(train\_images, train\_labels, numepochs, learning\_rate)  
  
# 在测试集上进行预测  
predictions = model.predict(test\_images)  
  
# 计算准确率  
accuracy = np.mean(predictions == test\_labels)  
print("Test Accuracy:", accuracy)

**运行截图:**

**文本

描述已自动生成**

1. **实验总结**

**实验过程中遇到的第一难题是不知道从何下手,神经网络概念还是比较抽象,最后从网上查阅了相关概念才有点头绪,其次就是MNIST数据库的调用也用了较长的学习时间,在模型的训练处理上用过很多python的库但是都不太熟练导致结果不理想,最后还是选择了numpy进行操作,最终完成了模型的训练.  
得到的结果是，随着训练的进行，损失值逐渐减小，模型的预测结果与实际结果之间的差异逐渐减小，最终训练后的测试准确率为95.098%，但是还是存有疑虑，因为每次的运行得到的最终准确率都不相同，时高时低，但总体稳定在95%附近。**