认知科学与类脑计算

实验报告八

日期：2019/5/31

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

一.需求分析：

BP 算法是由学习过程中信号的正向传播与误差的反向传播两个过程组成。由于多层前馈网络的训练经常采用误差反向传播算法，人们也常把将多层前馈网络直接称为 BP 网络。BP 算法由信号的正向传播和误差的反向传播两个过程组成。BP 算法的基本思想是，学习过程由信号的正向传播与误差的反向传播两个过程组成。正向传播时，输入样本从输入层传入，经各隐层逐层处理后，传向输出层。若输出层的实际输出与期望的输出(教师信号)不符，则转入误差的反向传播阶段。误差反传是将输出误差以某种形式通过隐层向输入层逐层反传，并将误差分摊给各层的所有单元，从而获得各层单元的误差信号，此误差信号即作为修正各单元权值的依据。这种信号正向传播与误差反向传播的各层权值调整过程，是周而复始地进行的。权值不断调整的过程，也就是网络的学习训练过程。此过程一直进行到网络输出的误差减少到可接受的程度，或进行到预先设定的学习次数为止。本实验加深对 BP 算法的理解，能够使用 BP 算法解决简单问题。

二.概要设计：

输入数据[[0.35], [0.9]]，真实值[[0.5]]，输出数据与真实值的误差使用平方和误差表示。创建人工神经网络。参数更新值Δw 用链式求导法则求出，最后使得输出数据与真实值的误差小于 0.01。

三.详细设计：

BP 算法的学习过程由正向传播过程和反向传播过程组成。在正向传播过程中，输入信息通过输入层经隐含层，逐层处理并传向输出层。如果在输出层得不到期望的输出值，则取输出值与期望值的误差的平方和作为目标函数，转入反向传播，逐层求出目标函数对各神经元权值的偏导数，构成目标函数对权值向量的梯度，作为修改权值的依据，网络的学习在权值修改过程中完成。误差达到所期望值时，网络学习结束。

反向传播算法主要由激励传播和权重更新这两个环节组成，通过反复循环迭代，直到网络对输入的响应达到预定的目标范围为止。

激励传播。每次迭代中的传播环节包含两步：前向传播阶段，将训练输入送入网络以获得激励响应；反向传播阶段，将激励响应同训练输入对应的目标输出求差，从而获得输出层的响应误差。

权重更新。对于每个突触上的权重，按照以下步骤进行更新：将输入激励和响应误差相乘，从而获得权重的梯度；将这个梯度乘上一个比例并取反后加到权重上。这个比例将会影响到训练过程的速度和效果，因此称为学习率。梯度的方向指明了误差扩大的方向，因此在更新权重的时候需要对其取反，从而减小权重引起的误差。

四.调试分析：

使用了两层的网络，但样本只有一个，所以该模型的拟合能力相对该问题来说非常强，所以推测分布较为复杂，解空间较大，使用不同的初始化验证该想法

多个初始化参数如下

```
w1 = np.array([[0.1, 0.8], [0.4, 0.6]])  # 第一层权重参数
w2 = np.array([0.3, 0.9])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.8, 0.1], [0.6, 0.4]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.4, 0.7], [0.1, 0.2]])  # 第一层权重参数
w2 = np.array([0.1, 0.1])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.4, 0.6], [0.8, 0.1]])  # 第一层权重参数
w2 = np.array([0.2, 0.4])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.1, 0.6], [0.2, 0.2]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.8, 0.7], [0.6, 0.2]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.8, 0.1], [0.6, 0.4]])  # 第一层权重参数
w2 = np.array([0.3, 0.3])  # 第二层权重参数
bp(w1, w2)
```

五.测试结果：

测试了七次，不同初始化下的 w1，w2 结果差异很大，但都可以很好的拟合

第一个为 w1，第二个为 w2，第三个为结果

num 1

[[0.10060672 0.75759502]

 [0.40156013 0.49095861]]

[[-0.29967374 0.32221246]]

[[0.50134267]]

num 2

[[0.75037818 0.10078759]

 [0.47240103 0.40202523]]

[[0.33700326 -0.30392033]]

[[0.50179664]]

num 3

[[0.39946344 0.69925194]

 [0.09862029 0.19807641]]

[[-0.00447181 0.01543179]]

[[0.50136949]]

num 4

[[0.39844141 0.58980494]

 [0.7959922 0.07378414]]

[[-0.10558101 0.12882789]]

[[0.50139194]]

num 5

[[0.0529996 0.59923602]

 [0.07914182 0.19803549]]

[[0.24631772 -0.26827529]]

[[0.50228828]]

num 6

[[0.76212964 0.69906787]

[0.50261907 0.1976031]]

[[0.22351299 -0.26087448]]

[[0.50136174]]

num 7

[[0.79371025 0.094426]

[0.58382636 0.38566685]]

[[0.01585308 -0.00611918]]

[[0.50136658]]

同时发现，w1 的结果与初始化的结果十分接近，可见对 w2 的调整更多，这是误差反向传播的时候 w1 的数值梯度较小，小于 w2

六.附录：

附录代码如下 文件为 bp.py

```
    import numpy as np

def sigmoid(x):  # 激活函数
    return 1 / (1 + np.exp(-x))

def bp(w1, w2):
    input = np.array([[0.35], [0.9]])  # 输入数据
```

```
    real = np.array([[0.5]])  # 真实值
    for s in range(0, 100, 1):
        pq = sigmoid(np.dot(w1, input))  # 第一层输出
        output = sigmoid(np.dot(w2, pq))  # 第二层输出,也即是最终输出
        e = output - real  # 误差
        if np.square(e) / 2 < 1e-6:
            print(s)
            break
        else:
            # 否则,按照梯度下降计算权重参数
            # 其中,应用链式法则计算权重参数的更新量
            w2 = w2 - e * output * (1 - output) * pq.T
            w1 = w1 - e * output * (1 - output) * w2 * pq.T * (1 - pq.T) *
input
    # 输出最终结果
    print(w1)
    print(w2)
    print(output)


w1 = np.array([[0.1, 0.8], [0.4, 0.6]])  # 第一层权重参数
w2 = np.array([0.3, 0.9])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.8, 0.1], [0.6, 0.4]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.4, 0.7], [0.1, 0.2]])  # 第一层权重参数
w2 = np.array([0.1, 0.1])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.4, 0.6], [0.8, 0.1]])  # 第一层权重参数
w2 = np.array([0.2, 0.4])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.1, 0.6], [0.2, 0.2]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)

w1 = np.array([[0.8, 0.7], [0.6, 0.2]])  # 第一层权重参数
w2 = np.array([0.9, 0.3])  # 第二层权重参数
bp(w1, w2)
```

```
w1 = np.array([[0.8, 0.1], [0.6, 0.4]])  # 第一层权重参数
w2 = np.array([0.3, 0.3])  # 第二层权重参数
bp(w1, w2)
```